3-23-2017

AutoProv: An Automated File Provenance Collection Tool

Ryan A. Good

AUTOPROV: AN AUTOMATED FILE PROVENANCE COLLECTION TOOL

THESIS

Ryan A. Good, Lt, USAF

AFIT-ENG-MS-17-M-031

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-031

AUTOPROV: AN AUTOMATED FILE PROVENANCE COLLECTION TOOL

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Ryan A. Good

2nd Lt, USAF

March 2017

AFIT-ENG-MS-17-M-031

AUTOPROV: AN AUTOMATED FILE PROVENANCE COLLECTION TOOL

THESIS

Ryan A. Good
2nd Lt, USAF

Committee Membership:

Gilbert L. Peterson, PhD
Chairman

Samuel J. Stone, PhD
Member

Alan C. Lin, PhD
Member

AFIT-ENG-MS-17-M-031

Abstract

A file's provenance is a detailing of its origins and activities. There are tools available that are useful in maintaining the provenance of a file. Unfortunately for digital forensics, these tools require prior installation on the computer of interest while provenance generating events happen. The presented tool addresses this by reconstructing a file's provenance from several temporal artifacts. It identifies relevant temporal and user correlations between these artifacts, and presents them to the user. A variety of predefined use cases and real world data are tested against to demonstrate that this software allows examiners to draw useful conclusions about the provenance of a file.

Acknowledgments

I owe a deep debt of gratitude to my advisor and committee members for guiding me through the rewarding process of creating a Thesis. Thanks also to Kevin Cooper, for providing excellent insights on several difficult coding concepts. Finally, thanks to my friends and loved ones for being there to support me and make my time at AFIT truly enjoyable.

Ryan A. Good

Table of Contents

List of Figures

List of Tables

1. Introduction

Computer forensics, which involves analyzing a digital medium for evidence of or related to a crime [8], requires the tracking and digesting of a myriad of files and their relationships. Parsing through this information is a daunting task, and the time requirement of this analysis can prevent an examiner from quickly obtaining the information they need to solve a crime. The relationship between files and their origins, along with the times and ways in which they were modified and accessed, and by whom, can help greatly speed up this process.

An object's provenance is its place of origin, or its history [23]. Based on this definition, file provenance, an important forensic resource and the inspiration for this study, is the "ownership and the actions performed on a data object"[15]. Ownership describes who created the file, or who brought the file onto the system, while actions describe post arrival file interactions. Automating the identification of these digital data relationships can help greatly expedite the forensic process.

1.1 Forensic Process

According to the Department of Justice, the steps in the forensic process are Preparation/Extraction, Identification, and Analysis [5]. Figure 1 shows these steps and their presence within the greater scope of the investigation.

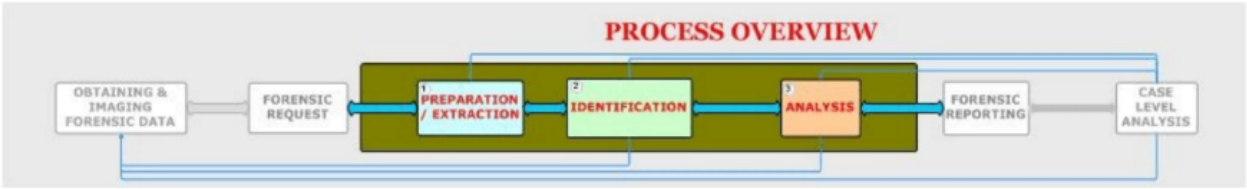


Figure 1: DOJ Forensic Steps [5]

When attempting to determine where a file originated from, an examiner starts by mounting the image they are interested in as read only. Mounting the image as read only ensures

the image integrity maintained, and that the data within is not altered in any way. This is important, because if the image is modified it is no longer viable in a court room setting. After the image is mounted, the examiner looks at the metadata for the file under investigation. The metadata contains a plethora of useful information, such as the creator of the file, who last modified the file, and when these things occur. It may be tempting to just take these values and assume that the origins of the file are known. Unfortunately, these values are often missing, and are easily modified by a savvy cyber criminal. Therefore, the examiner must either fill in the missing information, or validate whatever data is available.

The examiner should then view the `mtime, ctime,` and `atime` of the file they are interested in [17]. The `mtime` is the last time the file was modified. It is updated whenever the content of a file is changed. The `ctime` is also updated whenever a file's content is changed; however, it can additionally be updated whenever a file's attributes are changed. A file's attributes can be changed by a number of factors, such as file movement, or ownership change. The `atime` is the last time the file was interacted with in any way. This can be a result of simply opening the file. To summarize: if the file is simply opened and viewed, only the `atime` changes. If it is opened, viewed, and edited, the `atime` and `mtime` both change. If the file is opened, edited, and placed into another directory, then all three of these values change. It is also important to note that all three of these values change if a file is copied and pasted. This is because copying and pasting is creating a new file. If a file is simply moved this does not occur, as it is still the same file.

At this point, the examiner is free to pursue a number of paths, in whatever ordering they choose. There are a number of actions that most examiners normally turn to in order to determine the provenance of a file. The first is to use the tool log2timeline, in order to create a time line of events that occurred on the image of interest. The examiner could always construct a time line manually, but using a tool such as log2timeline is far more efficient. log2timeline provides the examiner with information on all the activity within the system.

Generating this time line; however, takes a great deal of time. Fortunately, the examiner most likely does not need a time line on all of the system's activities. Therefore, they can use filters to have the tool only run on the more relevant portions of the image.

Now that these time lines are available, the examiner begins to parse them in order to gain a more thorough understanding of the system. The most valuable information is within the system's registry. Time lines of individual user activity are within the NTUSER.dat hive, while the SYSTEM registry contains various system configuration information.

One of the richest sources of provenance on a file system is the registry [6]. The registry consists of a number of hive files that do anything from holding system configuration information to tracking the activities of individual users. Whenever something happens on the system, it is almost a guarantee that it impacts the registry in some way. All registry keys have a value called the "last write time". Whenever a relevant event occurs, the value changes. This is what time line generators such as log2timeline look to when they construct their time line of system events.

The registry consists of two separate types of hives: System hives, and User hives[6]. System hives include the Security, Security Account Manager (SAM), System, Software, and AmCache hives. User hives include NTUSER.DAT and USRCLASS.DAT. System hives associate with the overall functioning of the computer system itself, while user hives relate to specific users (each user has their own NTUSER.DAT and USRCLASS.DAT hives). For example, The security hive contains the system's operations, as well as data pertaining to configurations. The SAM hive stores user credentials, and enables user authentication both locally and remotely. The most valuable registry for file provenance is the NTUSER.DAT user hive, as it logs user activities, including file interactions and program executions.

1.2 Problem Statement

Based on the brief summary above of the path an examiner takes to rebuild a file's provenance on a forensic image, it is clear that this task is quite possible. Unfortunately, gathering all of this data and parsing it is time consuming, there is currently no software that automates the entire process. Software that can quickly rebuild the provenance of a file by noting correlations and generating time lines can save forensic professionals a great deal of time, and allow them to more quickly determine the origins and activities of any files of interest to an investigation.

Completing this task on a forensic image is far more difficult than on a live machine. A live machine provides access to the system's memory, and the ability to track all activity as it occurs. With a forensic image, the user faces the consequences of large amounts of time passing without the tracking of any system activity. During this time period, data availability can fade as it is automatically wiped from the system [4]. Information can also be purposefully removed, and this activity is much more difficult to discover on a forensic image. Access to the system's Random Access Memory (RAM) is lost as well, as all information stored within RAM is volatile. This includes data in use by applications that the user currently has open, which can contain valuable insights into the user's activities. Therefore, any automated discovery of provenance related information is restricted to the system's registry hives, various configuration files, and file metadata.

1.3 Hypothesis

It is possible to recreate the provenance of a file located on a forensic image, in most cases. This is true as long as enough information is available within the metadata of the file, as well as the registry. It is possible to automate the process used to gather this provenance information using forensic tools that are already available. This thesis creates a proof of

concept algorithm that automates the gathering of provenance, without creating software that is redundant enough to account for all possibilities or be fully and undoubtedly correct. The provenance provided by this algorithm requires verification by the examiner; however, the provenance is highly useful in tailoring the investigative process and saving time.

1.4 Assumptions

This thesis assumes that the files this tool analyzes do not have their provenance purposefully obfuscated. Dealing with the activity of tools such as Timestomp [24] is outside of the scope of this research project. This tool is useful as a guide to further tailor the investigative process, and is not intended to draw definitive conclusions about the provenance of a file.

1.5 Contributions

This thesis presents a proof of concept piece of software named AutoProv, short for Automated Provenance, that shows it is possible to thoroughly recreate the provenance of a file of interest on a forensic image. All previous provenance rebuilding efforts focused on dealing with live images, and tracking all user and program activities in order to rebuild provenance. AutoProv shows that, while not nearly as thorough as fully tracking provenance on a live image, it is possible to recreate large portions of the provenance of a file present on a forensic image. This is especially true when dealing with files that provide useful metadata, such as Microsoft Office documents.

1.6 Summary

A file's provenance consists of various forensic artifacts that enable the discovery of a file's history. This includes the file's creation, as well as any other movements and modifications enacted upon the file. Tools already exist that enable the automated gathering of a file's

provenance on a live system. This thesis shows that automated provenance gathering is also possible on a forensic image.

2. Related Work

The work related to this topic includes anything related to file provenance, and the gathering of provenance data. Following the discussion of provenance and its sources is a passage on the current systems that are in place to automatically track file provenance. These "file provenance maintenance systems" are constantly active, tracking all activities on the system in order to build a complete provenance picture. The discussion then focuses on studies related to automating forensic analysis without the use of a backbone system, such as a File Provenance Maintenance System (FPMS). A summary of the various tools discussed within this chapter is available within Table 1.

Table 1: Tool Summary Table

Tool Summary			
Provenance Maintenance Systems	**Image Analysis**	**Time Line Generation**	**Automation**
PASS	FTK	log2timeline	pyflag
FiPS	EnCASE	PLASO	ramparser
OPUS	TSK	PyDFT	FACE
		Zeitline	RegRipper
			Tapestry

2.1 Provenance

Provenance refers to the earliest known history of something. It can also refer to the record of ownership of an object [23]. Therefore it makes sense that when dealing with computers, provenance is the origins of a piece of data, its relationship to other pieces of

data, as well as the process that created it. Provenance data is extremely valuable to scientific forensic communities. It ensures that data is accurate, as well as current. There are many ways that files can obtain additional provenance data. A file created by one person could be edited by many others. It could be transferred between them using FTP, email, and many other protocols and methods. The source of some of its content may be from another file, which could be copied and pasted over. All of these factors contribute to the origins of a file.

One important part of provenance analysis is understanding the timing associated with the events surrounding the creation, modification, and transference of a file. Analyzing the history associated with a file provides excellent insights to examiners on the origins of that file. A simple starting point for learning about a file's creation and movements is time stamps. Time stamps exist in some form on most if not all file systems, and provide interesting insights into a file's modification history, or even its creation. Unfortunately, the recording of useful history such as last access time is disabled by many system power users. Tools such as Timestomp allow users to alter their time stamps and can complicate an examiner's attempts at tracking user history.

Provenance is also useful as meta-data, especially within scientific and business applications [15]. Meta-data can allow for much more powerful searches, enabling a user to search for a file based on who worked on it in the past, or its origins. Many times users forget the exact document that is of interest, but remember that they sent an email to someone about relevant data. With this information, it is possible to narrow the spectrum of possible documents, allowing the user to more quickly identify the object of interest. Correcting mistakes is another valuable use of provenance data. A user may create a file that contains data that influences other files. If that data is incorrect, it has far reaching consequences that the original creator is not aware of. Fortunately, with provenance, the creator of the original data can find out who else is using this information, and alert them to the correction.

A large part of forensic investigations involves searching for a particular piece of data

within a file. Many tools automatically search the system's storage for a string or other relevant information [2]. However, if the information is split between multiple locations, such as the registry and the browser history, the tool may fail. Therefore it is important for examiners to find out the search methodology of their tool of choice, and ensure that it is able to find information split between multiple data units. Unallocated data can also contain important information. Examiners often search unallocated data units to determine if there is any meaningful data present. TSK provides a tool, dls, that is able to accomplish this task. According to TSK, data not used by the file system is unallocated. All other data is allocated.

Technological changes inevitably bring with them more provenance acquisition challenges. For example, the advent of Near Data Processing (NDP) is an important change in computer architecture. In NDP "specific computations can be directly executed on the low frequency wimpy cores used in storage devices" [1]. This has interesting implications for provenance, as it could cause difficulties in tracking the changes to data that occur on these storage cores. Provenance completeness is important for the fidelity of any provenance system, and without this data the provenance is lacking. For this reason, computer hardware must become an integrated part of the provenance process.

2.2 Sources of Provenance Data

One of the first places an inspector looks for provenance data is a file's metadata. The metadata of a file contains information on the creation date of the file [29]. It also records the last modification date of the file, which is the last time a change was made. In many cases, especially when dealing with Microsoft Office, the metadata also records the user name of the individuals who create and modify the file. This information is valuable when trying to determine attribution.

Finding redundant data for verification purposes is a powerful solution to the potential

incorrectness of time stamps and metadata. Registry files are an excellent source of this kind of information, as well as browser history. Proxies and firewalls, entities that a suspect cannot interact with without advanced skills, are also valuable for determining the a file's provenance.

The registry contains a number of hives that each possess data on the various system and user operations [6]. For example, the security hive contains information on the audit policies of the system. The audit policies determine whether certain information is available, such as the history of users logging into the system. The SAM hive determines all the user and group configurations/accounts on the system of interest. The most useful information it presents includes the name and permissions of the groups and users present, the creation date and time of the accounts and groups, and the date and time of each user's last log in. Information on USB devices is also available within the registry, as the system hive keeps a log of USB activity. This information includes the date and time a USB device was last attached to the system. USB data is valuable when combined with other relevant system interactions. Through examining user activity during the time span of the USB's attachment, it is possible to determine what user account is associated with connection of the USB device to the system.

Web browsers are one of the most common file sources, as users use them as an interface to acquire various files from web hosts [19]. Email is another common file source, and many users forgo desktop email solutions such as Microsoft Outlook, and choose to use a web based solution from within their browser instead. Most browsers have a built in structure for maintaining their web history, which is non-volatile. This history is useful for determining the probability of a file arriving via web download, and the download source if this is the case.

Proxies and firewalls often keep activity logs. These logs are valuable for verification when an examiner is unsure of the validity of the data found within a system [32]. By cross-

referencing the information found within a firewall's logs with the information of interest on a system image, an examiner obtains greater certainty that the data is accurate.

2.3 Tools for Provenance Data

A variety of tools are available for gathering file provenance, and forensic data in general. The three most common ones used in the initial analysis of an image are FTK, TSK, and EnCase. EnCase requires much more training for the examiner than the other tool sets, searching is confusing, and there is no log file available for examiners [18]. However, EnCase has incredible search capabilities and allows for greater analytic convenience. FTK is easier to use than EnCase, requiring less training for the examiner. This is mostly due to its intuitive interface. Unfortunately it also has a lengthy image importing process and lacks customization options [18]. TSK, on the other hand, has a vast array of customization options thanks to Perl scripting, and works well with other Linux tools. Detracting from this, is its lack of ability to identify encrypted files, and vagueness in notifying the operator of existing overwritten files.

Log2timeline is a valuable tool for extracting temporal artifacts from digital media. The historical data contained within a forensic image, when gathered from the many sources that are available, is difficult to manage and parse effectively [12]. Log2timeline provides a framework that helps forensics experts view the image's history without becoming completely overwhelmed with all of the information presented. It also bypasses much of the work that comes with extracting time stamps from a variety of sources, as different artifacts store time stamps in different formats. Log2timeline aggregates all of these time stamps into a manageable format. It uses the same time stamp annotations as TSK: `crtime` is when the file was created, `mtime` is the time at which the file was last modified, and `atime` is the last time that the file was accessed.

Log2timeline excels at pulling all of the information from an image that has a time stamp

associated with it. Unfortunately, it does not allow for the aggregation of that data into a form that allows for easily determining the provenance of a specific file. For a user interested in a file's provenance, the data that log2timeline provides may not be entirely and obviously valuable. Therefore, it is necessary for someone interested in such data to apply filters in order to avoid extraneous information, as well as employing other tools to aid them in quickly finding the information of interest, such as a file's metadata.

A useful tool for automating the task of pulling any relevant metadata from a file is exiftool [13]. Exiftool is a Perl library and command line application, and allows the user to view and edit the metadata on a number of files. Due to its ability to modify metadata as well as read it, this tool is usable as a means of obfuscation to thwart examiners. It works on a plethora of metadata formats, including: EXIF, GPS, IPTC, and many others. If this tool fails to pull the relevant metadata from a file, the metadata most likely is not present. The metadata of a file may often lack the information an examiner is searching for. The system's registry is invaluable for filling in any gaps.

Harlan Carvey created a tool, `RegRipper` , that makes gathering data from different hive files within the registry much easier than manually parsing them[6]. For example, the `RegRipper` plugin `auditpol.pl` extracts audit policy information from the registry hive. Another useful pluggin, `samparse` , extracts user and group configuration/account settings from the SAM hive.

If the examiner is interested in USB devices, `RegRipper` allows for the extraction of relevant information from the system hive. The `usbstor` plugin allows them to determine the installed external devices, as well as the date and time at which this took place, identifying the device by serial number. They then use the mountdev plugin to tie the device's serial number to a drive letter, or even a device model name. The usbdevices plugin is also valuable for determining the last write times of the USB devices.

Other useful plugins give the user the ability to see various data related to the network

cards, and connections made on the system of interest. The networkcards plugin provides information on the networking interfaces present on the system at the time of image capture. If the wireless access points the system connected to are of interest to the investigation, then the ssid plugin obtains the SSID of the wireless access point (WAP), the MAC address, as well as when the system last connected to this WAP. If more in depth information is desired in regards to the WAPs, the networklist plugin is capable of filling in any gaps. It is important to note that the time zone is factored into these outputs, which often creates a discrepancy between the last write date/time and the last connected date/time.

An examiner needs confirmation to prove that a file originated from a web browser. It is possible to employ the histories of the various web browsers installed on the system to accomplish this. NirSoft produces a number of tools that enable quick and simple extraction of this information, such as the MZHistoryView [27] tool that enables the extraction of the Fire Fox browser's history. The NirSoft tools present the web sites visited, as well as the dates and times at which these visits occur. Unfortunately, this information is not presented in chronological order, and the user must remedy this manually.

When activity occurs within a system as a result of the presence of a malicious file, examiners often look to the network's firewalls for external verification of the data found within the system of interest. Keeping track of this information, and picking out data that is potentially malicious is a difficult task. A tool called VisualFirewall[16] attempts to facilitate examiner's efforts by providing visual representations of the various activity within a firewall. It provides four separate views that help narrow down the myriad of information present so that an examiner can prioritize resources towards what is important to their investigation. This data helps examiners learn about user interactions with the file post-arrival, or the file's original source.

2.4 File Provenance Maintenance Systems

There are many tools related to the tracking and gathering of file provenance. The goal of some of these tools is to create a system that monitors all file provenance on the host machine. These systems are known as provenance maintenance systems, and they run in the background of the system, doing what is needed to account for provenance data while requiring no input from the user besides their normal computer operations. For example, the Provenance Aware Storage System (PASS) "automatically collects, stores, manages, and provides search for provenance"[25]. In order to accomplish this task, it maintains provenance in memory and on disk. When dealing with a disk, the primary concern a forensics analyst has is the references files have between one another. In memory, elements such as pipes and sockets come into play and help determine how provenance is created.

Unfortunately, PASS has some shortcomings. It is unable to automatically collect opaque provenance, which is defined as data originating "from a non-PASS source, such as a user, another computer, or another file system that is not provenance aware" [25]. It also results in substantial overhead when dealing with large files. In one of the scenarios presented this processor consumption overhead was a large as 232.41%. Space consumption is also a major issue with this implementation, as simply deleting 24 KB of data resulted in 3486 KB in PASS overhead.

Sultana and Bertino[30] propose a system called FiPS as an alternative to the domain-specific approach used by PASS. In their words,"The fundamental problem with domain-specific approaches is that the data object and the provenance are managed by two separate data management systems". Not only does FiPS implement the functionality of PASS, it also allows for the re-creation of files created using undocumented methodologies. This is a very useful feature, as scientists and engineers often spend hours trying everything imaginable to solve a problem, only to quickly forget the steps taken to generate this solution. With FiPS in

place, these steps are recorded for later review and documentation, allowing for replication of the methodology. FiPS avoids the overhead associated with system call tracing, the strategy used in PASS, by enacting an implementation that places itself between the "Virtual File System (VFS) and any other file system which results in space and time efficiency" [30].

All of these systems provide valuable forensic data, and enable the quick determination of how a file arrived on a system, in addition to its relationship with other files. Unfortunately, these systems require installation on the host machine in order to enable them to keep track of provenance. They must be in place while the actions of interest occur, to allow for live observance of the relevant system calls and file interactions that allow for the creation of relationship and origin databases. This type of system is impractical when dealing with a system that is not within a corporate, government, or scientific environment. If a computer does not have one of these systems installed during the time of the incident, then there is no relevant evidence for examiners to use. Most information systems do not come with file provenance backbones pre-installed. Therefore, systems that allow for provenance gathering from a data source that did not have any assisting provenance software in place are essential for certain applications, such as criminal cases.

OPUS, or Observed Provenance in User Space, attempts to solve this by creating a provenance system that is able to be dropped into any existing system to capture provenance with minor additional overhead/complication[28]. OPUS also accomplishes runtime context collection, allowing for more detailed provenance models. Instead of just focusing on major file system operations like existing models, it captures all operations allowing for more detailed models. This is accomplished by intercepting and capturing provenance at the C library level through overriding the application symbol table. OPUS also introduces the Provenance Versioning Model (PVM). PVM provides a more formal view of important operations and allows for the abstraction of I/O semantics, which enables clear understanding despite differences in various operating systems. This type of functionality is important in a world

where many file system entities are accessible simultaneously.

There are many provenance systems available that track activity on a system in real time, ensuring that there is a thorough log of all activity within a system. It is important to note, however, that most of these systems do not try to ensure the validity of the provenance they capture through securing the provenance information. Securing provenance is a critical endeavor, as provenance information is often put to use in determining guilt or innocence in criminal cases. Researchers attempted to solve this by creating a provenance-aware system prototype that focuses on ensuring the validity of provenance [9]. In this study, the capture of provenance occurs at the application layer. Through thorough controls, the software ensures that no one is able to add or remove provenance information, preventing undetected rewrites of history. Trusted auditors verify provenance information, which is available in a format that makes it easy to determine if alterations are present. Data writes are put to use for the purpose of tracking this information, as they are far less computationally expensive to track than data reads due to the quantity present in day to day user operations.

2.5 Automating Digital Forensics

Many of the forensic tools discussed in Section 2.3 allow examiners to retrieve data from a disk image. However, these tools do not find evidence for the examiner. Instead, they retrieve all files and data from the disk, as any of it is potentially relevant. The examiner must then parse through all of this information, in order to find the pieces that are applicable to the investigation at hand. Tools that automate the time consuming process of finding pertinent evidence are extremely valuable. This is possible by locating files relevant to the investigation, optimizing data organization and presentation, and by creating a time-line of activity on the system.

One of the most difficult tasks for an examiner is determining what files within an image are of interest to the investigation at hand. The examiner must spend a great deal of time

parsing through potential sources of data, until finding something that is valuable. This task is more easily accomplished by examiners with more experience, as they are adept at finding other relevant files based on the data discovered thus far. Fortunately, tools are now available that facilitate this task, and help to decrease the time required to accomplish this process. Autopsy, or TSK, suggests additional searches based on items already marked as evidence[4]. These evidence markers help to create a target definition, stored as an object. Using these target objects, TSK searches for relevant evidence using various criteria provided by the examiner.

The examiner uses this to search for evidence that relates to the time field of the object, or the object's application type. Carrier [4] also developed a system that analyzes outliers in order to detect files that are out of place with the rest of the data. For example, root kits or other attack tools would be considered outliers. Discovering out of place files in this manner greatly aids the forensic examiner's ability to quickly process the data presented to them. The process behind discovering these outliers involves using file attributes to determine whether a file is abnormal or not. Unfortunately, Carrier himself states that since the multiple attribution method also results in some hidden files being missed, it is difficult to draw conclusions on the effectiveness of his method of outlier detection.

A 2010 study by Margo and Smogor [20] reinforces the notion that automated analysis is plausible. After finding that the semantic attributes necessary to perform advanced file searches were exorbitantly difficult to extract, they decided to use file provenance to find files placed in arbitrary locations. They were able to discover the relationship between files and processes by examining their location relative to one another, and the frequency with which they were accessed. This provenance data is fed into a machine learning algorithm that classifies the files by semantic attributes, including file extensions, through prediction. The results of the study showed that it was possible to predict the extension of a file based on its provenance data.

Data organization and presentation is an important part of the forensic process that greatly expedite or debilitate data gathering depending on how well it is implemented. One important facet of data presentation is the ability to generate a report on the information that is present. Pyflag has the ability to provide automated reporting due to the higher level analysis and extraction abilities provided by its scripting language, PyFlash [11]. This tool has already proven very capable, as it was the primary tool used in the Digital Forensics Research Conference (DFRWS) Forensic challenges in 2007 and 2008.

A more clear and concise view of relevant data makes is easier for specialists to quickly parse through information that aids in their investigations. Reduced workload for examiners, as well as more parsable information means that relevant evidence is found quickly. The FACE tool[7] does just this, as it does a better job of presenting the forensic data than most tools that are currently available. Its primary focus is correlating forensic data on the file-system with data within the memory, as well as the network capture. The ability to correlate data within the computer to information transmitted over the network is crucial, as it is easy to spoof source information, such as a MAC address. It also helps to prove that the computer the file was found on was indeed the original source of the file and not simply a recipient.

Ramparser, a tool for linux memory analysis is built to feed directly into FACE. It adds a number of features that were not available in automated forensics before the development of this tool. These include additional process functions such as identifying running processes, and how they were interacting with the stack. Ramparser also allows for identifying files that were open at the time of the image capture, as well as libraries that were shared. Lastly, it automatically collects information related to open sockets, and the related protocols that are being used to transmit data on those sockets.

Time lines are another visualization aid that make it easier for forensic analysts to understand what is happening on a system. It is possible to create time lines by hand; however,

it is far more efficient to have software available that automates the creation of these time lines. The advent of time line software began with a tool called Zeitline [3] in 2005. The purpose of Zeitline is to better organize the data present on a dead image. Encase, TSK, and FTK already gather this information, but the creator of Zeitline argues that it is not presented in a fashion that is useful for determining a sequence of events. Zeitline attempts to solve this by allowing examiners to import events, which are then grouped, filtered, and presented in a manner more indicative of a time line or sequence. log2timeline expands on Zeitline's functionality

Log2timeline is currently the most commonly used timelining tool, along with its powerful backend, Plaso. Plaso comes with 5 tools that are useful in forensic analysis. The first, image export, is useful for exporting file content from an image. It does this based on search criteria provided by the user, simplifying the image search process. The next tool it contains is log2timeline, which is the primary time line visualization aid. The third is called "pinfo", and it allows the user to get info from a plaso storage file, created by log2timeline.

Plaso files contain information on: when and how the tools was run, information gathered during the pre-processing stage, metadata about each storage container or store, what parsers were used during the extraction phase, parameters used, how many extracted events are in the storage file, the count of each parser, if there are tagged events, what tag file was used, what tags have been applied and count for each one, and if analysis plugins have been run, an overview of which have been run and the content of the report. The fourth tool is "preg", and is used to parse registry information off of a windows image. The last tool, "psort", enables automatic analysis on plaso file contents as well as sorting and filtering.

The information provided by log2timeline still has extraneous and overwhelming data that the examiner is not interested in. Due to this, any means of better organizing the data present is a welcome improvement. The data is presented in CSV format, allowing the user to open the document in excel to aid viewing; however, this does little to aid the informa-

tion inundation that the user is facing. A tool created by Derek Edwards called Tapestry attempts to remedy the situation [9]. Tapestry organizes data by month, day, year, or hour and hides extraneous data until the user wishes to expand upon it. This greatly helps with the information overload that occurd with log2timeline. Beyond these basic capabilities, the program also summarizes any changes in MACB records, which show access, modification, ownership, and creation times. This makes it easy to see any interesting behavior, such as file modification or movement. The program enables the user to create custom highlighted groups based on system activities such as USB mounting. These highlighted groups simplify parsing the information at hand for related actions. To top things off, Tapestry even summarizes the activities occurring within these groups.

2.6 Summary

A file's provenance is its history. This consists of the various users who owned the file, the times at which various interactions occurred, and the methods used to facilitate ownership and interaction with the file. Ownership and interaction with the file have a direct relationship. If a user edits the file, they now have ownership of it. An examiner interested in a file's provenance is interested in all of the ownership inducing interactions a file experiences, as well as more nuanced interactions such as simply transmitting or opening the file. This information is useful for facilitating anything from criminal investigations to the scientific process.

There are several sources of provenance data. The most common source associated with provenance is a file's metadata. Information is also available in the system registry, the various browser's installed on a system that the file was found on or traveled through, and proxies and firewalls. A number of tools exist that enable the retrieval of this information. FTK, TSK, and EnCase enable the mounting and creation of images, as well as data retrieval. log2timeline collects time stamped information from throughout the system, and aggregates

it into a manageable format. Exiftool pulls the metadata from files. `RegRipper` and its associated plugins allow users to easily access information within an image's registry. NirSoft provides a suite of tools that extract the history from browsers. Through combining these tools, an examiner is able to rebuild the provenance of a file.

The most commonly employed provenance tracking programs are File Provenance Maintenance Systems. These systems rely on live tracking of a system's activities. They record any interactions with the files on the system, building a thorough provenance record for all the files contained within. Due to their reliance on live tracking, as well as their dependence on the system's memory, they are not useful for rebuilding provenance for files located on a system image. Many tools available today automate forensic processes, but none of them enable the automated extraction of provenance from a dead source.

3. Digital Forensics File Provenance Generation

This section first describes AutoProv in broad terms, giving the reader an overall idea of its structure. Following that, it provides a more technical and in depth explanation of the software. This begins with the overall structure of each major piece of software used in the design: DataGather.py, and DataProcess.py, both written in python. Within each of these sections, the overview elaborates on specific pieces of data and the processes involved in gathering and parsing them.

3.1 Overview

There are many tools currently available to aid examiners in determining activity within system images. This software simplifies analysis by gathering information for the examiner, and providing it in a useful and readable format. Unfortunately, there are no software packages that automate the process of using these tools to aid the examiner in determining where a file has come from, and what activity is directly related to the file. This thesis presents a proof of concept tool that provides an initial prognosis of what enabled the file to arrive on the system as well as partial activity of the file post arrival. The software details the reasoning behind these provenance theories to the examiner, along with all of the tool outputs leading to this conclusion. The examiner is then able to use this information to determine where they should look further to verify the findings of the software.

The AutoProv system, shown in Figure 2, first runs a suite of tools in different configurations, and outputs the results of these iterations into text files. A folder encapsulates these text files, using sub-folders for information that pertains to particular users found within the image. Once these tools are run, by executing DataGather.py and providing it with the image and file of interest, DataGather.py creates the folder for later use. The examiner then runs DataProcess.py on the folder DataGather.py creates. DataProcess.py looks at all

of the data within this folder, and attempts to determine the origins of the file of interest within the image. It accomplishes this by looking for indicators that an examiner often uses to determine the origins of a file. These origins could include local creation, a web browser, a USB drive, etc. The software informs the user of the indicators present, and they draw their own conclusions and determine how to best tailor their analysis from that point on. DataProcess.py inserts relevant pieces of this information into time line format to aid the examiner in prioritizing data analysis. The examiner then verifies this information by looking at the data DataGather.py provides, as well as whatever other sources are necessary for verification purposes.

In order to facilitate the readability of the software, both DataGather.py and DataProcess.py use a shared library called AutoLib.py. This library contains functions both DataGather.py and DataProcess.py use. The DataGather.py and DataProcess.py scripts are primarily a series of function calls, and the utilization of those calls. This provides a general functionality map to the reader. If the reader wishes to obtain a more in depth understanding of how the code is accomplishing its objectives, AutoLib.py contains the details of what each of the functions is doing.

The output this software provides is correct the majority of the time, as long as the necessary data is salvageable from the dead image. For example, if the Created Date is not salvageable from the metadata of a file, this affects the accuracy of this software's prediction. In this situation, more analysis by the examiner is necessary to verify the results. If enough pieces of data are missing, the software's output is not representative of the actual origins of the file. This tool is meant to be an aid, not a substitute for an examiner. Figure 2 provides a summary of the structure of AutoProv.

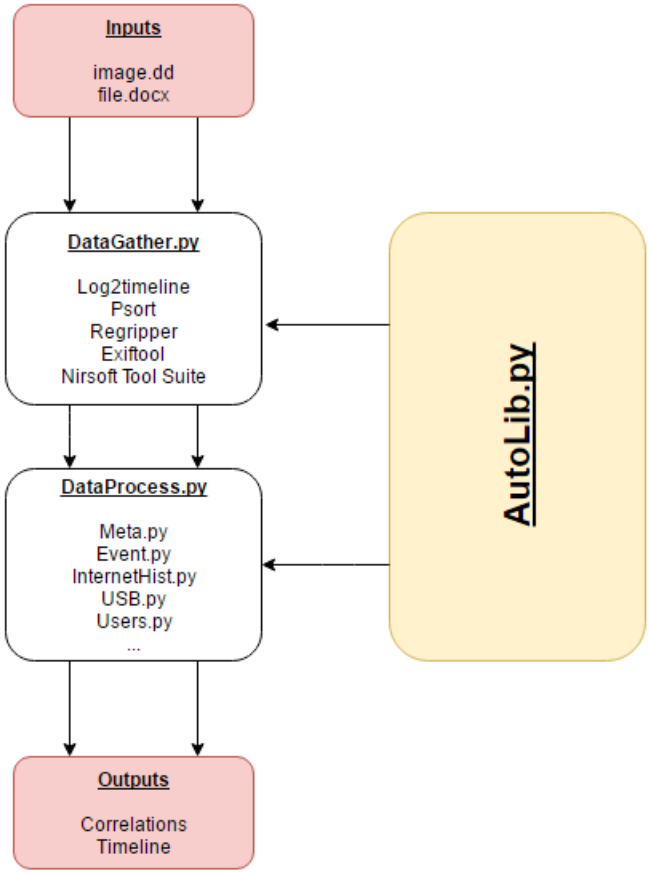


Figure 2: AutoProv Project Structure

3.2 Data Gathering (DataGather.py)

The data gathering component of this software occurs within one script: DataGather.py. The purpose of this script is to automate the many tedious tasks the examiner must often undergo when attempting to determine the provenance of a file. It also ensures that all of the data gathered is in a directory format that DataProcess.py is expecting, and therefore able to process and use effectively.

The various different software iterations result in different granularities of time data. Some of the results are within microseconds, while others are days apart. A summary of this information is available in Table 2.

Table 2: Temporal Granularity

Target	Source	Granularity
Registry last modified times	NTUSER.dat	microseconds
Recent Documents	NTUSER.dat	> days
File MAC times	File of Interest	seconds
History Entries	Browser History Files	seconds
USB Key	SYSTEM	seconds
User/Group Information	SAM	seconds
CurrentVersion subkey	SOFTWARE	seconds

After installing the required software and making the recommended path configuration modifications, the examiner need only supply the script with the location of the image, along with the name of the file of interest. At this point, the Data Gathering software mounts the image locally as a read only drive. DataGather.py mounts the image as read only, to ensure that neither the software nor the examiner accidentally modify any of the data/files present on the image. Ensuring the mounted image is read only also prevents any changes to last accessed times. This helps maintain the validity of the results for use in a court room setting.

To mount the image, the Data Gathering software needs to know the start block of the image of interest. It obtains this information by running `mmls` on the image, and routing the output to a text file. The text file is then parsed to find the start block of the first NTFS partition within the image. Simply looking for the string "NTFS" within a line in the `mmls` output allows the program to find the start block of the NTFS partition. DataGather also ensures that a start block for an NTFS partition has not already been found, as this would result in that partition's start block being overwritten with the second partition's start block. The first partition is the one of interest, as this contains the registry hives and user directories. DataGather assumes the block size is 512, as this is almost always the case by tradition. Once this information is gathered, the "mount" command is used to locally mount the image.

Once the image is mounted, DataGather then determine the location of the file of interest. In order to accomplish this, the find command is run on the image folder (the place where the image was mounted). The results are stored in a variable, which is then read and stripped of white space. The file's name and location are then stored within a text document for later use by DataProcess.

After discovering the location of the file, DataGather.py obtains the operating system version and installation date. This is useful for tailoring tools to a specific operating system version. It also allows the examiner to know that a file's creation date is completely out of scope for having possibly been created on the image it is residing on. This is accomplished using the `winver` plugin for `RegRipper` . The data of interest is contained within the SOFTWARE hive. While `winver` does contain the operating system version, DataGather.py simply looks at the directory structure leading to the SOFTWARE hive. Based on this directory structure, it is able to determine what version of Windows is running. If the structure looks like: `%SystemRoot%/System32/config/SOFTWARE`, then the software knows it is dealing with a Windows 7 or later operating system. On the other hand, a structure of `%SystemRoot%/system32/config/software` means it is dealing with Windows XP. The boolean variables representing Windows 7 and beyond, and Windows XP, are set to true or false depending on the operating system version, in order to properly direct other programs that interact with the registry.

With the location of the file itself ascertained, the program seeks to determine if there might be a torrent version of the file. Whenever a user downloads a file using torrenting software, the user must first download a torrent file, which aids the torrenting software in determining how to download the file of interest. This torrenting file is usually just the name of the file with ".torrent" attached to the end. For example, if we were torrenting a picture named "flowers.jpg", the torrent file would be called "flowers.jpg.torrent". Therefore, in order to determine if a torrent file related to the file of interest is present on the system, all

that DataGather.py needs to do is run the find command on the file of interest's name, with ".torrent" appended to the end. If a file is found, then there is a very high probability that the source of the file was torrenting software such as FrostWire.

One of the most useful sources of file provenance information is a file's meta data. This is the next piece of information DataGather obtains. The exiftool software accomplishes this, as it is able to glean metadata from nearly any file type, if that information is available. Therefore, all DataGather needs to do to obtain the metadata is to run exiftool on the file of interest. This information is routed to a text file for later use by DataProcess.

After obtaining the file's metadata, DataGather pulls the dates and times relative to the times the file was interacted with on the system. This information is contained within the `ctime` `atime` and `mtime` [17]. These dates/times are compared by the processing script to gain a great deal of information about the file, and it is therefore important to pass this information along. To gather this information, the gather script simply runs the "stat" command on the file, and outputs the results to a text file. The results are in an easily parsed format, clearly labeling the information of interest.

Now that the metadata has been captured, we are interested in obtaining the last write times of any registry keys that may be useful, as well as any pertaining values. This is done using log2timeline, along with various filters. The filters are used to restrict log2timeline, preventing it from analyzing the entire image. While this does reduce the information available to DataProcess.py, it is necessary to greatly reduce the time necessary to analyze an image. The majority of information relevant to the provenance of a file is found within %USERPROFILE%/NTUSER.dat, as well as the various folders and sub-folders within the Application Data folder (%APPDATA%), and the folders and sub-folders within Local Settings, so not much, if anything, is lost due to this restriction.

Each user's NTUSER.dat file contains a plethora of data showing their activities on the system. The application data and local settings folders contain a variety of user specific

settings and configuration files. These often contain information on when file's are accessed by various executables within the system. Due to the file structure being different on modern machines (Windows 7+) than it is on Windows xp, the filter contains accommodations for the different operating system file structures in the form of "or" statements, to ensure that log2timeline is directed to the correct location.

It is also important for the examiner, as well as the tools being used, to be aware of all the users present on a system. DataGather either checks the Users folder or the Documents and Settings folder, depending on the version of Windows, to determine what users are present on the system. The program is not interested in the default users that are present on all systems, so those users are filtered out along with unrelated directories. This includes: "All Users", "desktop.ini", "Default", "Default User", "Local Service", "NetworkService" and "Public". Once this list of users is placed within an array structure, all of the users of interest are printed to a text document so that DataProcess is aware of their presence.

In order to better separate information as it pertains to each user on a system, each user is given their own sub directory within the overarching directory dedicated to each iteration of DataGather. The first thing that is placed into each of these user directories after they are generated is the documents and drives they have most recently interacted with. This is accomplished using the `RegRipper` plug in: recentdocs_tln. This plugin is run on the NTUSER file of each user present on the system.

The next thing DataGather obtains is the system's USB connection history. This information includes the first date/time that the system interfaced with the USB device, the most recent time the system interfaced with the USB device, and the serial number of the device. The date's and times of the various USB connections are valuable for determining whether or not a USB device was the source of the file of interest. Especially in situations where the file is malicious, if a USB device is the file source, additional usage won't occur. Therefore it is reasonable to assume that the last connection date of the USB device would

be reasonably close to the date that the file was first seen on the system. This information is obtained using the usbdevices plugin for `RegRipper` . It is contained within the %SystemRoot%/System32/config/SYSTEM hive.

Another highly important piece of information gathered from the system is the user and group data. This data includes when each user account and group were created, the respective permissions of each of those entities, as well as the last time each user logged into the system. This information is highly valuable when attempting to find who allowed the file to get to its current location, and how it originally arrived on the system. The information is gathered by running the `RegRipper samparse` plugin on the SAM hive of the registry.

An important piece of information to facilitate the parsing of the log2timeline logs is the timezone that the image uses. log2timeline gathers information in GMT/UCT, while some other sources of forensic information are relative to the system time. In order to correct this imbalance, the timezone that the system was in prior to the image being taken must be known. Luckily, `RegRipper` has a plugin that looks inside of the SYSTEM registry hive in order to find this information. Therefore all DataGather must do is run the `timezone` plugin on the SYSTEM hive, and output the results to timezone.txt, and DataProcess handles the rest.

3.2.1 Browser History

The next item of interest is the user's web history. File's are commonly downloaded from web pages, and therefore it is important look at the various web pages a user has visited. DataGather begins by processing Chrome's history. This is done using a piece of software aptly named "ChromeHistoryView" [26]. DataGather locates the chrome history file, assuming it is in its default location, and provides it to ChromeHistoryView. ChromeHistoryView is designed to be run on the Windows OS, so Wine is used to allow it to run in the Linux environment. The output of this program is pushed into a text file for later use by DataPro-

cess. Similar methods using various Nirsoft tools allow for the attainment of the rest of the various browsers' history data.

The Nirsoft browser history extraction tools work exceptionally well for the majority of browsers. Unfortunately, their Firefox history extraction tool, MZHistoryView [27], did not perform as well as was necessary for the successful implementation of this thesis. For this reason, a Firefox history extracting python library was added to the scope of this thesis. The library uses SQL calls to navigate through the various tables present within the places.sqlite file that contains Firefox's user history for versions 3 and beyond. There are two tables within places.sqlite that are of interest to this thesis, for the purposes of extracting a sequential history of the user's browsing. These are the moz_historyvisits and moz_places tables, which have the structure shown in Figure 3.

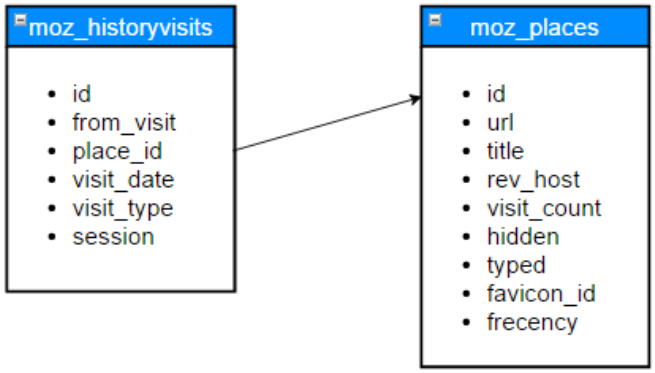


Figure 3: Fire Fox Table Relationship

Figure 2 shows the relationship between the two tables of interest within places.sqlite. There are many more tables within this database, but these are the primary two of relevance. Moz_historyvisits contains the sequential Firefox history of the user. The actual websites they visited are encoded as a place_id. This place_id is used as a reference to an entry within moz_places that contains the url of website visited. The moz_historyvisits table also contains the full date and time of the visit. This information is encoded in epoch format [10].

FFhistory.py converts this information into a array of history ojects that contain the address visited, the date of the visit, and the time of the visit. This is accomplished using the sqlite3 python library, which allows python to make SQL calls to an SQL table. The SELECT command is used to pull all of the information off both tables, which is returned in array format. The information of interest is then pulled using simple calls to the portions of the array that are of interest. Once the date/time of the entry is obtained in epoch format, the time library is able to convert it into GMT, a much more useful format. The various entries within moz_places are searched for each of the relevant indexes found within moz_historyvisits, and the addresses discovered are added to the relevant entries. By the time all of the entries found within moz_historyvisits are accounted for, the program has produced a full browsing history based on the user's places.sqlite database file.

3.3 Data Processing (DataProcess.py)

The data processing script receives all of the information it needs from a folder created by the data gathering script. Figure 4 shows this relationship. Once the data gathering script has run, all the examiner needs to do is execute the data processing script on the folder that was created by the gathering script. Once the script is done analyzing the various tool outputs produced by DataGather.py, it outputs any correlations as well as a time line showing many relevant events that helped determine their validity.

There are a number of boolean variables that DataProcess.py uses to track correlations it wishes to inform the user of. These variables and their meaning are overwhelming, so a summary is available at the end of the section. The time line is built using a doubly linked list. The nodes within the list are a class structure called an event. The event structure contains the year, month, and day of the time line entry. It also contains the hour, minute, and second of the event's occurrence. Any further granularity is removed from the event. Lastly it contains the description, which is the message printed to represent the event, and

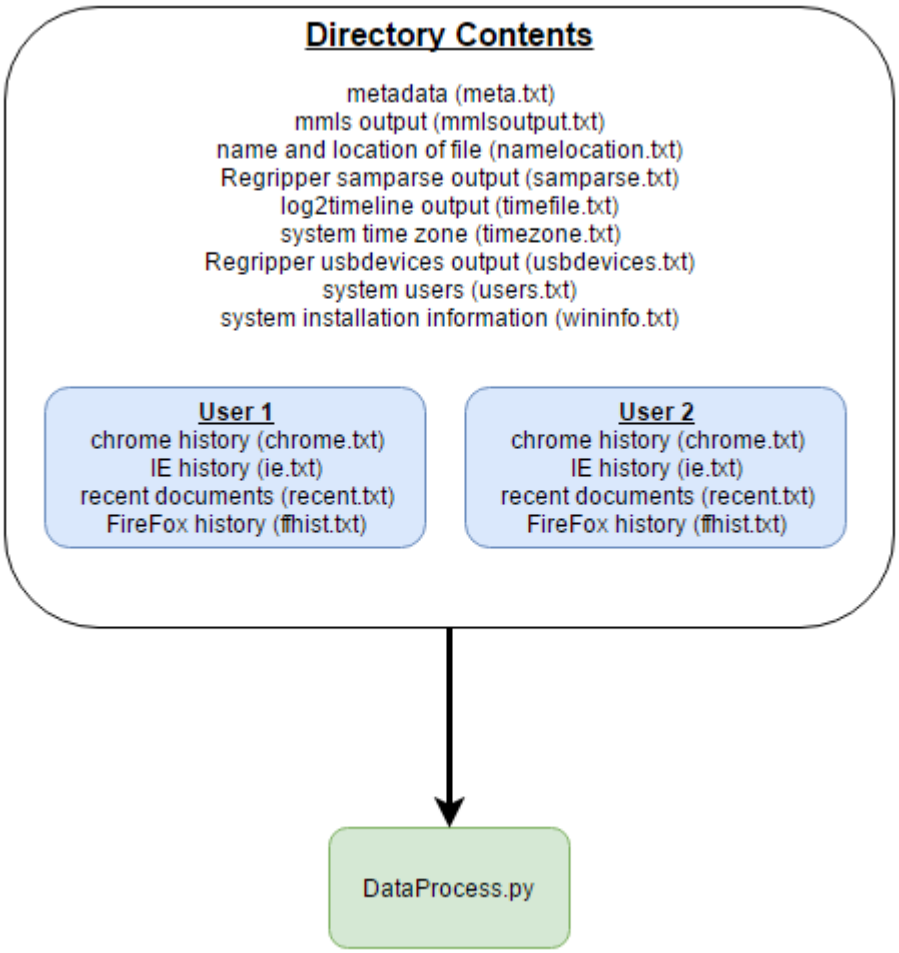


Figure 4: Output Directory Structure

pointers to the next and previous event nodes.

The first thing that the data processing script does, is open up the text file namelocation.txt, provided by data gather, that contains the file of interest's name and location. It then parses the contents of this file into two respective variables, filename and location, for later use.

The program also checks namelocation file to determine if a torrent version of the file exists on the system. If a torrent file exists, DataGather places the location of the file inside of the namelocation.txt file for the purpose of convenience in parsing. If a torrent file is not

found, then the third line of the namelocation text file is empty.

Next, the script opens the text file, meta.txt, containing the metadata of the file. This data is more difficult to parse, as the format is not always predictable. Based on the data that exiftool is able to find, it changes the output it produces. Therefore, instead of simply going line by line through the text file, the parse_meta function looks for specific strings within each line that are indicators that the information is present. Each line is stripped of extraneous blank space, and then split using colons as the parameter. This leaves a number of extraneous entries in the resulting array, which are removed. The function is then free to look for the strings "Creator", "Last Modified By", and "Create Date".

Through locating these strings, the function detects that some piece of the information of interest is present in the line. More blank space is stripped away from this line of value and it is split using the space character as a modifier. Unnecessary characters are removed. DataGather.py retrieves individual values of interest from the resulting array. It modifies any dates and times into a standard format used for comparison purposes with the dates and times produced by other tools. The format used for dates is year:month:day and the format for time is hours:minutes:seconds, with any extraneous zeros removed. For example, if a minutes value is represented by "05" minutes, it becomes "5" minutes for ease of comparison and casting to integers. With the metadata parsed, the script is free to determine if the file's creator is the same user who edited the file. If this information is available, and the values are different, then it is known that a different user modified the file than the user who created the file. This information is reported to the user.

The file's date of creation as well as the date of modification, are valuable for the time line this tool produces. At this point, the tool takes the information from the metadata, and parses it into the generic Event class structure format, enabling its addition as a node to the linked list time line. A function within AutoLib.py, insertEvent, takes care of adding these nodes chronologically to the time line.

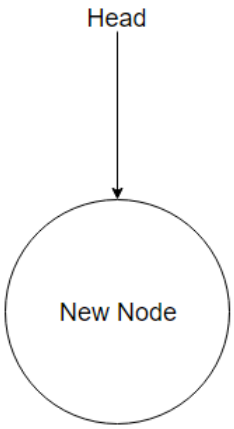
When nodes are added to the time line, insertion must occur at the proper position within the linked list structure. The insertEvent function accomplishes this by first checking if the list has a head node. If there is no head node, the new entry becomes the head of the list. Otherwise, the function iteratively checks whether each event in the list is greater than the new event. The isGreater function accomplishes this comparison by iterating through the values within the event structure, starting with year and ending with seconds, to determine which date and time is more recent. If the new event's date and time is greater, then the function places it after the current comparison node, and the isGreater function returns true allowing this to take place. If isGreater is true, the function checks the next node in the same manner as the previous one unless there is no next node. If there is no next node, the new node becomes the tail end of the list.

If the new node occurs before the node with which it is compared, then it is inserted before the comparison node. To accomplish this, the "next" pointer of the node before the current comparison node points at the new node, and the comparison node's "prev", for previous, pointer also points at the new node. The new node's prev and next pointers then point at these two nodes respectively. Using this insertion criteria, a chronological time line of events forms. Figures 5 and 6 show how this insertion method works.

Now that the metadata and the basic location information of the file are obtained, more detailed system information is desired. In order to obtain this information, DataProcess looks to the wininfo.txt file. As mentioned in the Data Gathering section, this file contains information on what type of operating system we are dealing with, as well as the date that the operating system was installed. DataProcess parses this file by calling the parseWinInfo function from the AutoLib library. This function grabs the information of interest, and reformats it to the same format as the other date/time values it is compared to.

First the function replaces the months with their respective numbers. For example, 'Jan' must be replaced with the number 1. The function uses a dictionary to accomplish

Empty List



Most Recent Entry

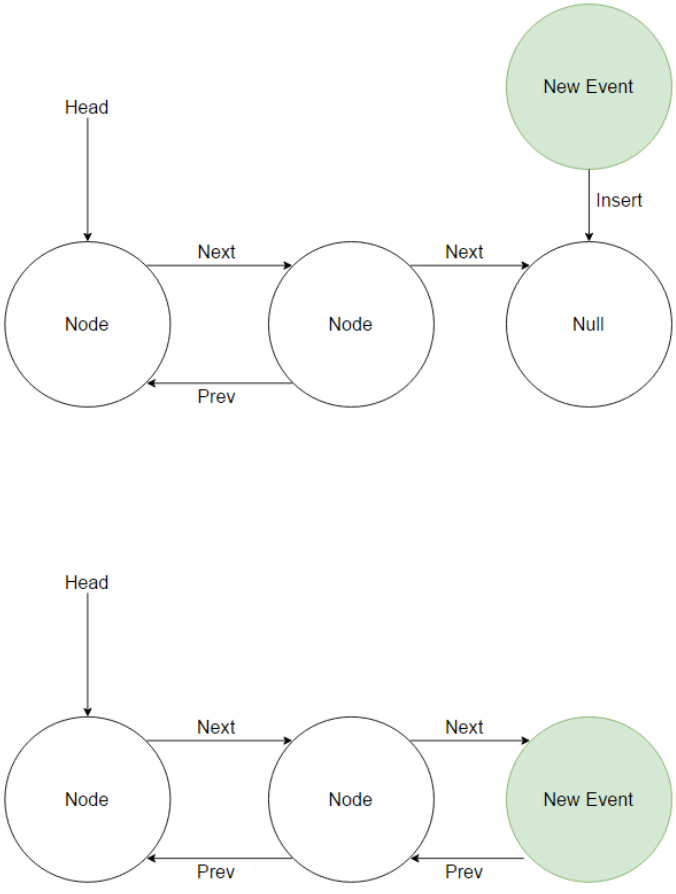


Figure 5: Time Line Insertions

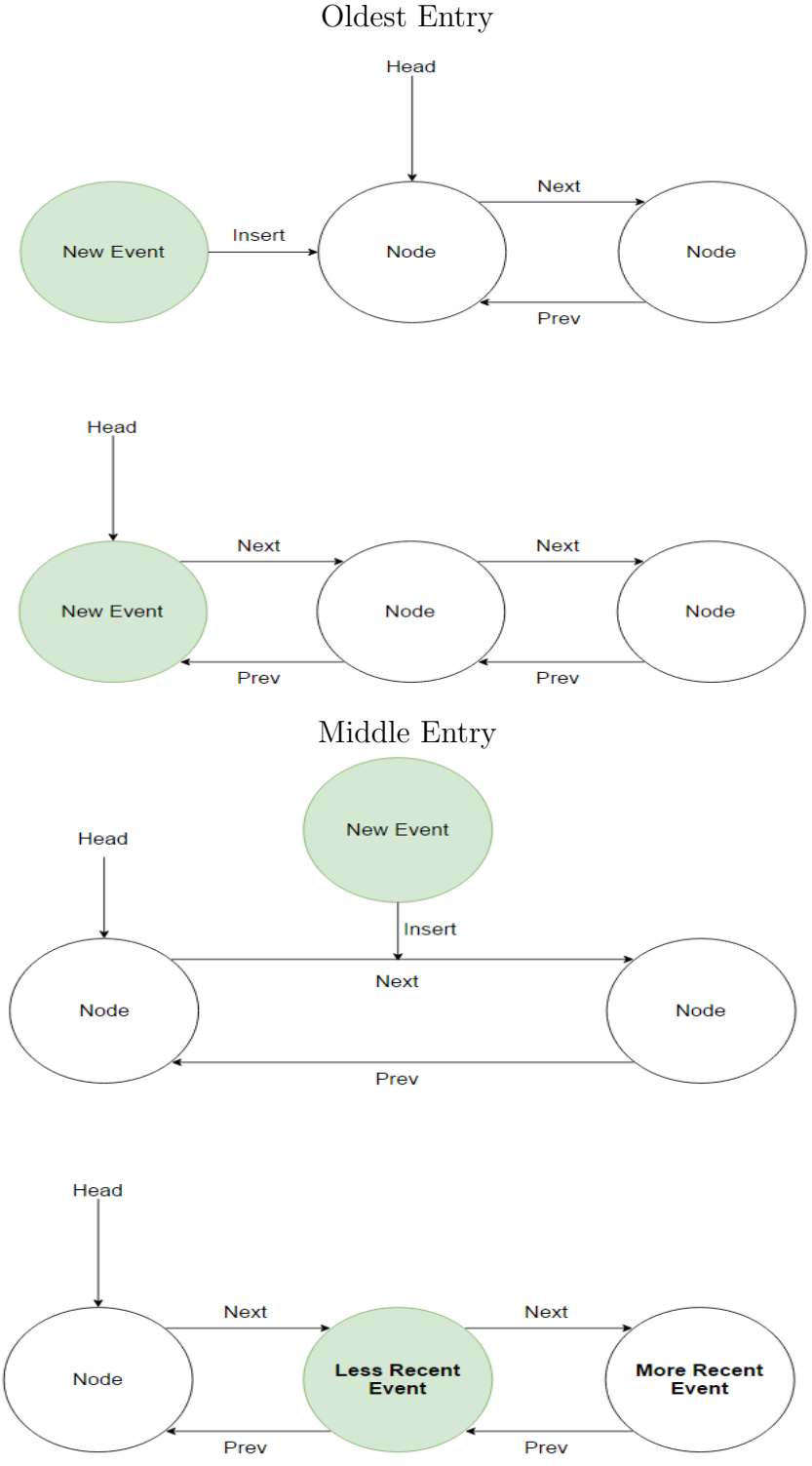


Figure 6: Time Line Insertions Continued

this. The three pieces of information the function pulls from the file is the "ProductName", which is the version of windows, the "CSDVersion", this is the last update applied, and the InstallDate, which is self-explanatory. This information is all stored within a WindowsInfo class structure, so that the information of interest is easily pulled whenever necessary.

Now that we have the metadata for the file, we look to see what users are present on the system. This information is held within the users.txt file generated by DataGather. DataGather has already done the majority of the work for us here, so all that is necessary is to iterate through the list of users, stripping off the new line character and placing each user name into an array. The first thing we use these user-names for is to determine if any of the users on this system are the user present in the "created by" section of the metadata. If this is true, a boolean is set that ensures the examiner is aware of this correlation.

The next items of interest are the `ctime`, `mtime`, and `atime`. This information is already present in timestamps.txt, thanks to DataGather. DataProcess pulls this information from the text file, and places it into the Timestamp class structure so that relevant information is easily available. Once again, the structure of the timestamps require alteration so that they match those of the other software in use. The parsetimestamps function within AutoLib does the heavy lifting here, with assistance from the timestampfix function. parsetimestamps searches for relevant strings that let the software know the information of interest is present, while timestampfix alters the times and dates found so that they are comparable to the times and dates presented by the other software in use. This, once again, requires replacing characters, removing blank space, and removing extraneous zeros from the minutes and seconds fields. The `ctime` is added to the time line as a reference point to show the user the file's creation date/time within the greater context of other relevant events.

AutoProv then examines the time line that DataGather provides from the output of log2timeline run with a filter pointing it at NTUSER.dat, as well as various application

folders, for the various users. Each user has their own set of directories/hives of interest. The DataGather script uses a filter for log2timeline that checks every user's useful structures, and outputs the results to a consolidated text file. Therefore, in order to find anything relevant within all of these entities, the script need only look within one text file. DataProcess looks for any entries that involve the file of interest within the generated text file, and appends them to an array of "relevant entries".

Whenever a relevant entry is found, its details are placed within an iteration of the time lineEntry class. This class structure provides the date of the entry, the time of the entry, and many other descriptive elements generated by log2timeline. log2timeline's output facilitates this, as it is in CSV format. Therefore, the script need only separate the values by comma, and strip away any excess characters, in order to populate the time lineEntry class values. The first relevant entry is then added to the time line that is provided to the user, so that the user sees when the file first arrived on the system within the greater context of the other relevant events.

Following the discovery of all relevant time line entries, DataProcess checks to see if any of them contain references to program executions on the day of the file's first arrival on system. This is done by looking for the first relevant time line entry that involves the file, and comparing the data of its occurrence to all time line entries that involve program executions. Program executions are discovered by looking for references to the userassist registry within the NTUSER registry in the time line report. The userassist logs are updated whenever a program execution occurs, making it possible to find program executions within a relevant time span. After finding all program executions that occur within the day of the file's arrival, DataProcess then narrows its criteria to programs executed within six hours of the file's arrival. The time line entries referencing these program executions, and the user's who executed the programs, are pulled and parsed into entries which are then inserted into the time line created by DataProcess.

This process is repeated for any program executions in close proximity to the file's modification time. This is important, as the file may have been modified within the system. If the file was modified within the system, any program executions that may have enabled the modification of the file become an important part of that file's provenance

The next step the script takes is to search for when software of interest was run by the user. The first software it checks for is FrostWire. FrostWire is a torrenting platform that is used to obtain files, and therefore it is within the scope of file provenance investigation. There are many other torrenting platforms, but for the purposes of this proof of concept, only FrostWire is detectable. Once again, the log2timeline results are used to determine if or when the software was run.

Within the log2timeline results, references to the userassist hive [31] show that the user ran specific software. Knowing this, the script need only look for a line containing a reference to "Frostwire.exe", as well as "winreg/userassist" to determine that the time line entry is a reference to the user running FrostWire at a particular date and time. The script then populates an array containing all entries that involve a FrostWire execution. We now know every occurrence of a user executing FrostWire.

These iterations of FrostWire executions are only valuable within the context of the file's arrival on the system. To determine if a FrostWire execution is of value, the DataProcess.py script checks if any of the executions occurred on the date of the file's arrival. If they did, then a relevant boolean variable is set to true, and the event is added to the time line in the usual manner, further adding to the user's understanding of the file's method of arrival.

Another commonly used application for file transfers is Skype. Skype allows two individuals to call one another, and also transfer file's during the conversation if they wish to do so. For this reason, DataProcess.py checks for Skype executions in the same way it checks for FrostWire executions. It looks for "Skype.exe" and "winreg/userassist" in the same line of the time line file. If it finds a situation where this is true, it logs the date and time of the

event. It then compares the date and time of the Skype execution to the file of interest's arrival data. If the dates match, a flag is activated. If the events are within thirty minutes of one another, another flag is activated, further refining the possibility of the file's arrival via means of Skype.

This proof of concept focuses mostly on the provenance of Microsoft Word documents. For this reason, it is important to note the activity of Microsoft Word. Similar to Skype and FrostWire, executions of WINWORD.EXE found inside the userassist subhive help determine when Word executions occur. Word executions are once again correlated with the arrival of the file; however, Word also allows users to create and edit files as well. For this reason, Word executions are also compared to the creation date found within the metadata, if a creation date is available. Separate flags are activated depending on whether the file arrived on the same date as a word execution, or if it was created on the date of the word execution.

The software also checks for the string "Microsoft/Office" within the display name of each time line entry. Within Windows XP, any uses of Microsoft office products contain this string. This helps with determining who interacted with the file of interest, and when they did so. The string "Removable Disk" is also checked for within the message portion of each relevant time line entry, as this highlights USB usage within close proximity to the file's arrival. This happens when the USB device and the file of interest both appear in the user's recently used documents.

Another important string to check for in the message section of relevant time line entries is "Content.IE5". This string existing in the message portion of a relevant entry means that a time line entry that references the file of interest, also references the content.IE5 folder, meaning that a reference to the file was found within this folder. This reference occurs due to the file being acquired using Internet Explorer, and based on the results discussed on the next chapter, it also appears to result from file's being acquired using Safari on Windows

operating systems. This immediately and greatly narrows the possible sources of the file of interest.

Besides knowing what happened to a file, it is important to know who caused the various actions to occur. For this reason, DataProcess also checks what users interacted with the file of interest. This is accomplished by searching the relevant time line entries for any references to a user that is known to exist on the system. If the user name is found within the display_name portion of the entry, and the user has not already been discovered, then the user name is appended to the list of users who have interacted with the file of interest.

It is also important to determine whether or not it is feasible for the file to have been created locally. If the creation date of the file is available within its metadata, the program compares this date to the installation date of the operating system, obtained by DataGather.py using `RegRipper`. If the installation date of the operating system is beyond the creation date of the file, there is no way that the file was created on the system, and therefore it must have been obtained via some other means. DataProcess.py also compares the creation date of the file to the first time the file is seen within the system of interest. If the creation time is within 30 minutes of the file's arrival on system, this is reported as this has a high correlation with the file having been locally created. This information helps drastically reduce the user's investigation efforts for rebuilding the file's provenance.

Next, DataProcess.py looks more into determining which users interacted with the file of interest. The program already obtained a list of users that are referenced within the output of log2timeline in conjunction with the file of interest. This has a high likelihood of accurately predicting who has interacted with the file. In order to further reinforce this knowledge, DataProcess.py looks at each user's recent interactions, as obtained by DataGather.py using `RegRipper`. If any of the recent interactions reference the file of interest, this information is reported to the user. This serves the purpose of further solidifying the validity of known information.

Another valuable piece of information for determining the source of the file of interest is the data on various USB connections within the system. This information was pulled via `RegRipper` by DataGather.py. DataProcess.py takes this raw text information, and parses it into a USB class structure. The date and time of the USB's last connection, as well as its serial number and device identifier are all obtained and parsed into an easily referenceable class format. If any of these USB devices were connected on the day that the file arrived, a correlation boolean is activated, and this information is reported to the user. The date and time of the USB connection is also added to the time line in order to better put this information in perspective.

DataProcess.py now attempts to further identify what users have knowledge of the file's presence on the system, and may know more about how it arrived. This is determined by looking at what users were logged into the system on the day that the file first arrived. This information is obtained from the user and groups information pulled from the SAM hive by DataProcess.py using `RegRipper` . The date that each user last logged in is compared to the date of the file's arrival on system. If the dates are a match, an entry is added to the time line and the user is appended to an array of users reported at the end of the program's execution.

The next section of DataProcess.py focuses on pulling all of the relevant web history off of the image of interest. It starts by checking if DataGather.py was able to pull each user's chrome history from the image. For each user that has available chrome history, each entry within their history is parsed into a class structure. Those class structures are then appended to an array of chrome history entry class structures. A similar method is used to pull any Internet Explorer history that is available for the system's user's. Nirsoft's Internet Explorer history viewer also pulls all of the various windows explorer interactions for each user as well, which helps with rebuilding the file's provenance. Each of these class structures is appended to an array.

Once the various internet history sources are parsed into class structures, the relevancy of the individual entries must be confirmed. In order to accomplish this, DataProcess.py splits the time of the first entry referring to the file of interest, and pulls out the hour of its occurrence. The program uses this value to narrow the list of web visits to within two hours of the file's arrival. This helps to ensure that the browsing history the user is viewing is relevant to the file's provenance. The program accomplishes this for each browser type sequentially, adding the relevant entries to the time line for later output.

3.3.1 Outputs

After all of this information is gathered, DataProcess.py begins outputting it in a user readable format. The first items of interest provided to the user are the correlations. All correlation flags are attributed to a meaning, summarized at the end of this section. The basic meaning of these correlations is conveyed to the user via a string output that highlights the occurrence of these items of interest. Arrays that are used as truth indicators lead to additional lines of output for the various users within the system. Once these boolean correlations are resolved, the program provides the user with the time line created throughout the various information filtering that took place earlier in the program's execution. This time line is referenced when the user wants clarification on the presented correlations, or when additional information is necessary to acquire accurate file provenance. The correlations presented to the user consist of several categories, shown in Figure 7.

Boolean Correlation Variables

The code example below each variable is simplified psuedo-code, and not the exact code used to activate the corresponding boolean.

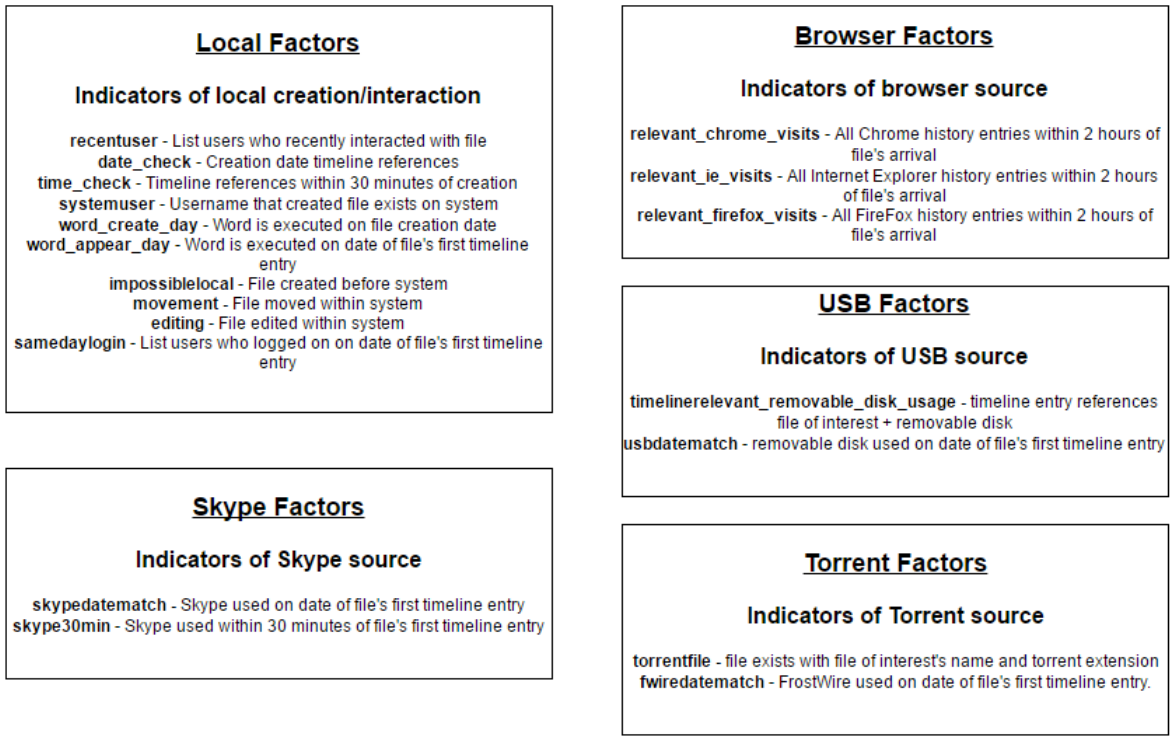


Figure 7: Correlation Categories

difmod

```
if metadata.creator != metadata.modifier:
  difmod = True
```

recentuser

```
for user in userlist:
    recent = recentdocs\_tln output
    for line in recent:
        if filename in line:
           recentuser.append(user)
```

date_check

```
if time line_entry.contains(filename):
    relevant_entries.append(entry)
for entry in relevant_entries:
    if entry.date == metadata.created_date:
        date_check = True
```

time_check

```
crtime = metadata.created_time
if time line_entry.contains(filename):
    releveant_entries.append(entry)
for entry in relevant_entries:
    if date_check:
        if (entry.time > crtime - 30 minutes) and
           (entry.time < crtime + 30 minutes):
              time_check = True
```

systemuser

```
for user in userlist:
    if (metadata.creator == user) or (metadata.author == user):
        systemuser = True
```

word_create_day

```
if time line_entry.contains("WINWORD.EXE"):
    word_entries.append(entry)
for entry in word_entries:
    if entry.date == metadata.created_date:
        word_create_date = True
```

word_appear_day

```
if time line_entry.contains("WINWORD.EXE"):
    word_entries.append(entry)
for entry in word_entries:
    if entry.date == file_arrival_date:
        word_appear_date = True
```

word_modify_day

```
if time line_entry.contains("WINWORD.EXE"):
    word_entries.append(entry)
for entry in word_entries:
    if entry.date == metadata.modification_date:
        word_modify_date = True
```

officeinteract

```
if time line_entry.contains(filename):
    relevant_entries.append(entry)
for entry in relevant_entries:
    if entry.contains("Microsoft/Office"):
        officeinteract = True
```

impossiblelocal

```
windows_info = winver output
windows_install_date = windows_info.install_date
if windows_install_date > created_date:
    impossiblelocal = True
```

editing

```
if mtime > file_arrival_date:
    editing = True
```

timelinerelevant_removable_disk_usage

```
if time line_entry.contains(filename):
    relevant_entries.append(entry)
for entry in relevant_entries:
    if entry.contains("Removable Disk"):
        time linerelevant\_removable\_disk\_usage = True
```

usbdatematch

```
devices = usbdevices output
for device in devices:
    if device.last_write_date == file_arrival_date:
        usbdatematch = True
```

torrentfile

```
if filename + ".torrent" on system:
    torrentfile = True
```

fwiredatematch

```
if time line_entry.contains("FrostWire.exe" and
                            "winreg/userassist"):
   frostwire_entries.append(entry)
for entry in frostwire_entries:
   if entry.date == file_arrival_date:
       fwiredatematch = True
```

skypedatematch

```
if time line_entry.contains("Skype.exe" and "winreg/userassist"):
   frostwire_entries.append(entry)
for entry in frostwire_entries:
   if entry.date == file_arrival_date:
       skypedatematch = True
```

skype30min

```
if time line_entry.contains("Skype.exe" and "winreg/userassist"):
  frostwire_entries.append(entry)
for entry in frostwire_entries:
  if date_check:
      if (entry.time > arrival - 30 minutes) and
         (entry.time < arrival + 30 minutes):
            time_check = True
```

samedaylogin

```
users = samparse output
for user in users:
    if user.last_login_date == file_arrival_date:
        samedaylogin = True
```

relevant_chrome_visits

```
chrome_visits = ChromeHistoryView output
for link in chrome_visits:
    if link.visit_date == arrival_date:
        if (link.visit_time > arrival - 2hrs) and
           (link.visit_time < arrival + 2hrs):
               relevant_chrome_visits.append(link)
```

relevant_ie_visits

```
ie_visits = IEHistoryView output
for link in ie_visits:
    if link.visit_date == arrival_date:
        if (link.visit_time > arrival - 2hrs) and
           (link.visit_time < arrival + 2hrs):
               relevant_ie_visits.append(link)
```

relevant_ff_visits

```
ff_visits = FFhistory results
for link in ff_visits:
    if link.visit_date == arrival_date:
        if (link.visit_time > arrival - 2hrs) and
           (link.visit_time < arrival + 2hrs):
               relevant_ff_visits.append(link)
```

userinteract

```
if time line_entry.contains(filename):
    relevant_entries.append(entry)
for entry in relevant_entries:
    for user in users:
        if entry.contains(user.display_name) and  (not userinteract):
            userinteract.append(user)
```

systemuser

```
for user in users:
    if (metadata.creator == user.display_name) or
       (metadata.author == user.display_name):
           systemuser = True
```

systemmod

```
for user in users:
    if metadata.modfier == user.display_name:
        systemmod = True
```

4. Experimental Results

Two sets of tests validate the functionality of AutoProv's ability to recreate provenance on an image. The first tests, Use Case tests, run through a set of manually generated scenarios. The second set of tests involve images where the provenance of the file's are unknown prior to running the AutoProv scripts. The tool set is run, and then the results are analyzed to determine there efficacy. The time zone of all dates and time referenced is GMT.

4.1 Use Case Testing

The first set of tests involve a system that is reset to a known good snapshot between tests. During each test, the user takes deliberate actions that the software adequately responds to. These tests successfully test the core functionality of AutoProv, without introducing the unexpected. The time line functionality was not yet implemented as of these tests.

4.1.1 Use Cases

Each use case involves a Windows 7 Service Pack 2, 64 bit, VMWare Virtual Machine with Google Chrome v. 51, Firefox v. 46, FrostWire v. 1.7.3, Microsoft Office 2016, and Skype v. 7.25 installed. The six use cases are:

1. A user logs on, creates a Microsoft Word (MW) document, and logs off. Another user then logs in, edit, and saves the document.

2. A MW document is copy-pasted to the system via NTFS format USB removable drive. A user on the system then edits and saves the MW document.

3. The user calls someone on Skype, and receives a MW document from them. The file is then moved.

4. The user torrents a MW document using FrostWire, and then edits the MW document.

5. A user downloads a MW document via Chrome, and then edits it.

6. Downloads a MW document using Internet Explorer, and then edits the document.

4.1.2 Use Case Results

Use case one resulted in the following boolean flags:

- **difmod** - The file was modified by someone other than the creator.
- **localuser** - The file is in one or more user's recent documents.
- **time_check** - Relevant time line entries exist referring to the file within thirty minutes of its creation.
- **systemuser** - The file's creator has the same user name as a user who exists on this system.
- **word_create_day** - Microsoft Word is used, within the system, on the creation date of the file.
- **word_appear_day** - Microsoft Word executes on the first day that this file is seen on the computer.
- **editing** - This indicates possible file editing while on the system of interest.

From these flags, the examiner easily determines that the file was edited, and that the editing most likely occurred locally. This is based on the editing and difmod flags, and supported by the local user flag. In addition, the time_check, systemuser, localuser, and word_create_day flags show the file was locally created.

Use case two resulted in the following boolean flags:

- **usbdatematch** - A removable disk is used on the same day that the file first arrived on system.
- **time linerelevant_removable_disk_usage** - This variable is set if a time line entry references the file of interest, as well as a USB drive.
- **word_appear_day** - Microsoft word executes on the first day that this file is seen on the computer.
- **difmod** - Creator and Modifier are different.
- **samedaylogin** - User's last logged in on day of file's arrival. Users are listed.
- **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

The usbdatematch and time linerelevant_removable_disk_usage combine to show that the file could have originated from a USB device. The word_appear_day and difmod flags then show that the file was most likely modified after arrival, using Microsoft Word. The samedaylogin flag is active, which means that any users who logged in that day are listed, helping to narrow down the user who connected the usb device. The time_check flag is active due to testing, as the file was quickly transferred after creation for the purposes of this test. Therefore, its presence is ignored.

Use case three resulted in the following boolean flags:

- **skypedatematch** - Skype was used on the same day that the file first arrived on the system.
- **skype30min** - Skype was used within 30 minutes of the file first being seen on system.
- **samedaylogin** - User's last logged in on day of file's arrival. Users are listed.

For the third test, the skypedatematch and skype30min flags trigger to show that Skype is used within 30 minutes of the file's arrival. The samedaylogin then lists the user who was logged in on the date the file arrived, helping the user to discover who allowed the file to arrive on the system.

Use case four resulted in the following boolean flags:

- **torrentfile** - A torrent file exists that has the same name as the file of interest.
- **fwiredatematch** - FrostWire was used on the day that the file was first seen on the system.
- **difmod** - The file was modified by someone other than the creator.
- **editing** - This indicates possible file editing while on the system of interest.
- **samedaylogin** - User's last logged in on day of file's arrival. Users are listed.
- **impossiblelocal** - The Operating System was installed on this system after the file was created.
- **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.
- **word_appear_day** - Microsoft word executes on the first day that this file is seen on the computer.

The fourth test works as expected, activating the torrentfile, and fwiredatematch flags to show a possible torrent source. The combination of difmod, editing, time_check, and word_appear_day flags create a high likelihood of local editing. The user safely assumes none of these flags were activated by the file's creation due to the impossiblelocal flag's activation.

Lastly, the impossible local creation flag activates. This lets the user know that the creation date of the file is earlier than that of the operating system. This dramatically decreases the likelihood of local creation.

Use cases five and six resulted in the following boolean flags:

- **relevant_chrome_visits** - This variable is true if there are any chrome visits within $+-2$ hours of the file's arrival on system.
- **relevant_ie_visits** - This variable is true if there are any Internet Explorer visits within $+-2$ hours of the file's arrival on system.
- **difmod** - The file was modified by someone other than the creator.
- **editing** - This indicates possible file editing while on the system of interest.
- **samedaylogin** - User's last logged in on day of file's arrival. Users are listed.
- **impossiblelocal** - The Operating System was installed on this system after the file was created.
- **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

The last two use cases have similar results, as they both test browser history parsing. The relevant_chrome_visits and relevant_ie_visits flags are both active for their respective tests, so the program provides the user with any web page visits within a two hour time span. This provides users with insight into the source of the file, especially in this test, as the file's source web page contains the name of the file itself on the download mirror. The tool presents the information in a readable single line format, allowing the examiner to easily parse the results. The usual login (samedaylogin) and editing (difmod, editing) flags are active as well, showing the user the file was modified locally.

4.2 Real Data Corpus (RDC) Testing

This portion of the testing uses a series of relevant images from the Real Data Corpus (RDC) [21]. The RDC consists of real world forensic data collected from devices purchased on the secondary market throughout the world. After mounting the images and locating files of interest, the data gathering and processing tools were used to recreate the provenance of the files. User names are changed to a numerical standard in order to protect identities. User enumerations are reset for different images (user 1 on image 1 is a different user than user 1 on image 2). The time line portion of the project was complete at this point, allowing for more complete provenance recreation.

DataGather.py uses correlations and filtering to reduce log2timeline's output from an average of several million events to fifteen events or less. This is shown in Figure 8.

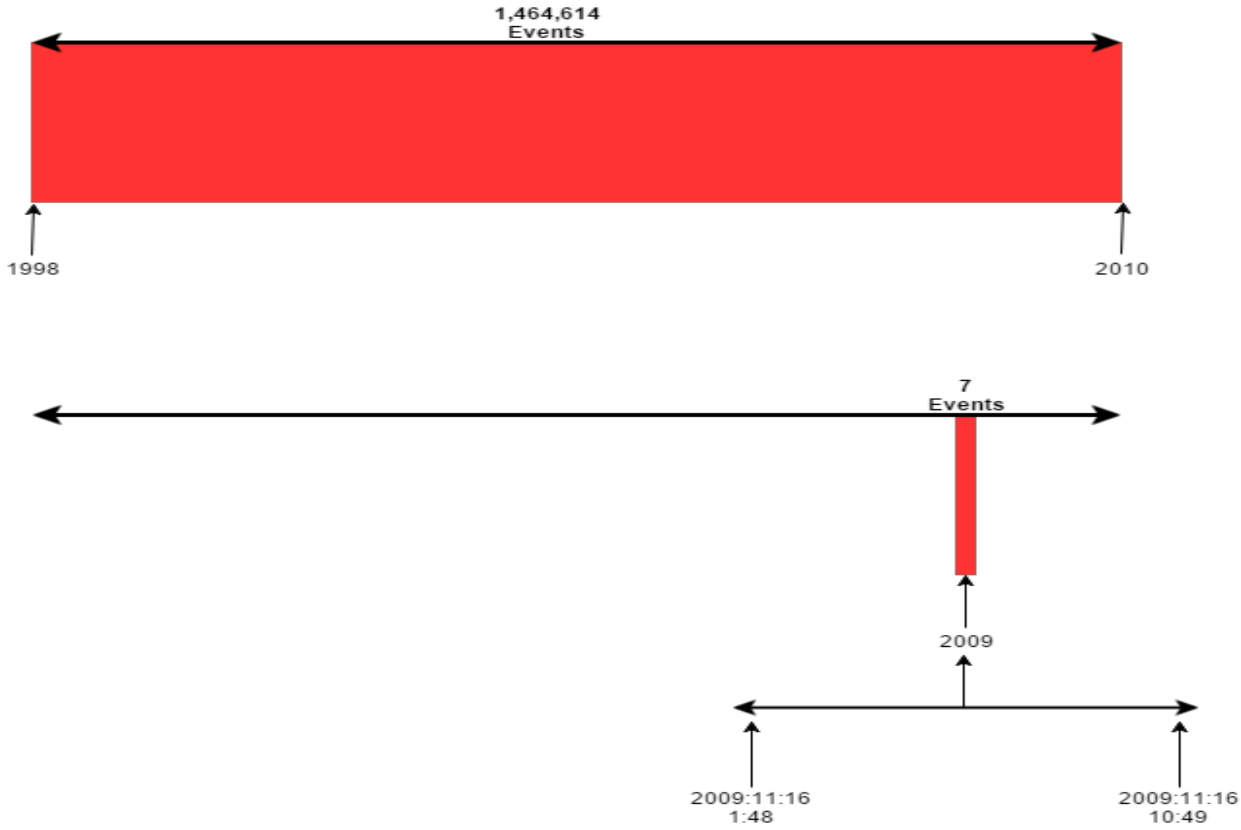


Figure 8: Time line Summary (Image 5: File 2)

4.2.1 Results

Image 1: File 1

The following booleans are set to True in for the first file of interest, a .doc (Microsoft Office) file, on image 1. Image 1 is a Microsoft Windows XP machine with Service Pack 1 installed.

- **Creator not Modifier** - Looking at the file's metadata, the software sees that the file's creator is not the same user as the file's modifier. This is not a boolean flag, but the software still outputs this information. The modifier's name is also provided.
  File Modified: 2006:6:19 - 08:36

- **datecheck** - This boolean is false, as the creation date is not equivalent to the date the file is first seen on system, resulting in the program noting that it is very unlikely the file was created locally. The file's creation date, and the date it is first seen are both provided to the user.
  Creation: 2006:6:8 - 06:34 // Arrival: 2006:6:20 - 11:41:38

- **userinteract** - A user, user 1, interacted with the file of interest.
  Arrival/First Interaction: 2006:6:20 - 11:41:38

- **officeinteract** - Microsoft Word was used to interact with the file.
  Arrival/First Interaction: 2006:6:20 - 11:41:38

- **editing** - The file appears to have either been edited or copied within the system due its mtime being much different than its arrival time. The mtime is provided to the user.

- **ie5_content** - References to the file exist within Internet Explorer's Temporary Internet Files (TIF).

– The time line includes entries that are not within a time span that infers local creation.

Figure 9 shows a summary of the time line produced by DataProcess.py.

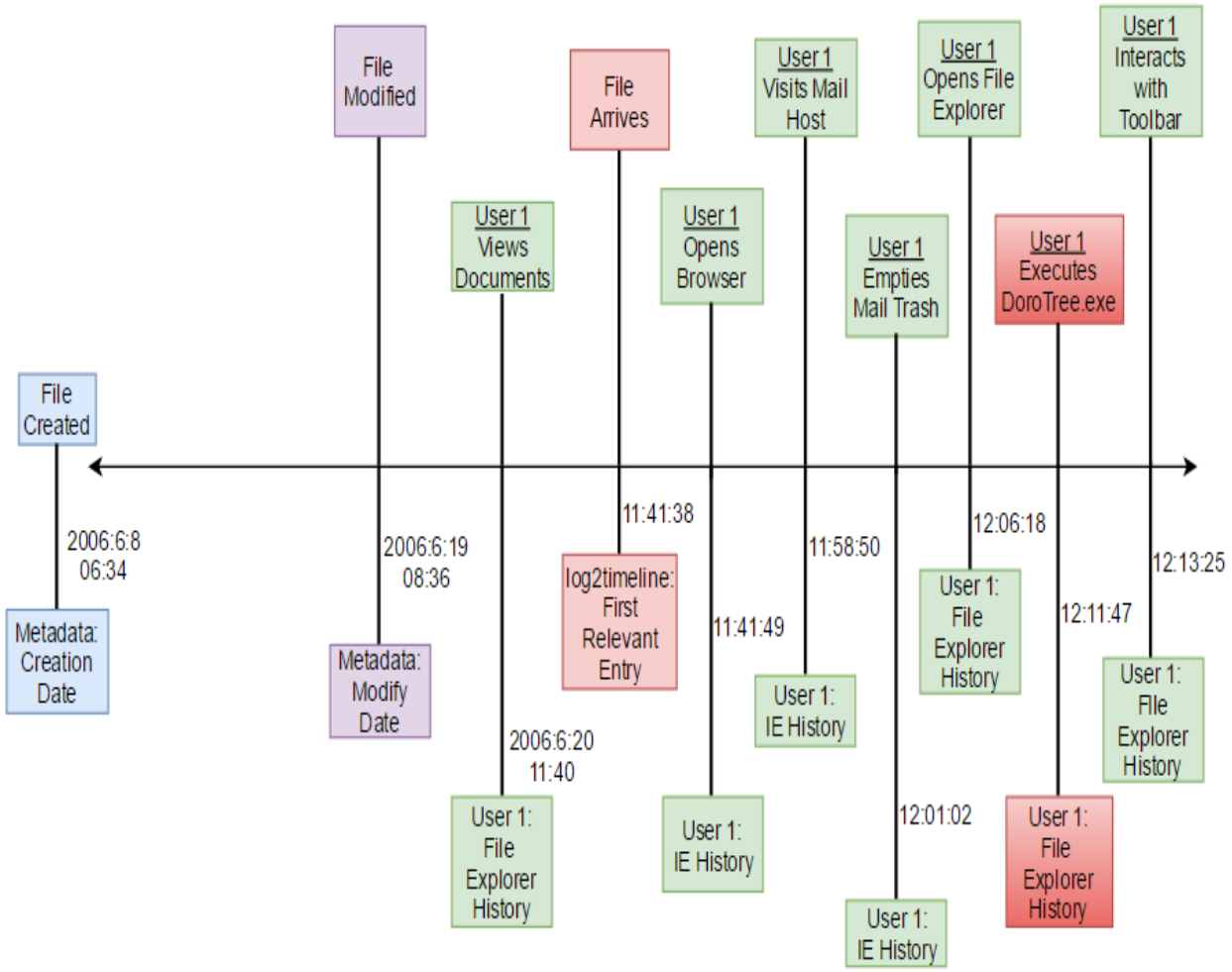


Figure 9: Time line Summary (Image 1: File 1)

Viewing the correlations determined by DataProcess.py, along with the time line is produces, it is fairly simple to recreate the provenance of this file. It is apparent based on the datecheck flag that the file probably wasn't created locally. Combining that with the fact that there are references to the file within Internet Explorer's content, it becomes clear that the file was most likely downloaded. The file was edited at some point based on the creator not being the modifier, but it probably didn't happen on this system, as the most recent

modifier is not a user on this system. The editing flag is most likely true due to the file being copied and pasted out of the downloads folder. User 1 also appears to have used Microsoft Word to view the file of interest, based on the userinteract and wordinteract flags, while not having edited it due to this user not being the last modifier.

The time line helps clarify this even further, as there is an entire day between when the file was last modified and its arrival on the system. The time line also shows that the user opened their browser and navigated to their web mail page as soon as the file arrived. This leads to the conclusion that the file was most likely collected via web mail, and then the web history was deleted. Considering the web history resumes 11 seconds after the files arrival, this is a fair conclusion to make. The user also appears to delete their web mail's history, further reinforcing the conclusion that they were attempting to cover up their tracks.

Provenance Narrative

Someone creates the file on another system, and User 1 downloads it via a web mail host on 2006:06:08 at 11:41:38 using Internet Explorer. User 1 then deletes their web mail history. Next, they move the file from the downloads folder to its current location. At some point after arrival, it is not clear when, they also view the file using MW.

Image 2: File 1

The following booleans are set to True in for the first file of interest, a .xls (Microsoft Excel) file, on image 2. Image 2 is a Windows XP machine with Service Pack 2 installed.

- **Creator not Modifier** - Looking at the file's metadata, the software sees that the file's creator is not the same user as the file's modifier. This is not a boolean flag, but the software still outputs this information. The modifier's name is also provided.

- **datecheck** - This boolean is false, as the creation date is not equivalent to the date the file is first seen on system, resulting in the program noting that it is very unlikely

the file was created locally. The file's creation date, and the date it is first seen are both provided to the user

Creation: 2005:12:31 - 08:22 // Arrival: 2009:11:16 - 10:39:45

- **userinteract** - A user, user 1, interacted with the file of interest.

  Arrival/First Interaction: 2009:11:16 - 10:39:45

- **officeinteract** - Office was used to interact with the file.

  Arrival/First Interaction: 2009:11:16 - 10:39:45

- **impossiblelocal** - The Operating System was installed on this system after the file was created.

  Creation: 2005:12:31 - 08:22

- **editing** - The file appears to have either been edited or copied within the system due its mtime being much different than its arrival time. The mtime is provided to the user.

- **usbdatematch** - A removable disk was used on the same day that the file first arrived on system.

  USB Connected: 2009:11:16 - 01:48

- **skype30min** - Skype was used within 30 minutes of the file first being seen on system.

  Skype Execution: 2009:11:16 - 10:27:51

- **samedaylogin** - User 1 and User 2 have a last log in date that is the same day the file arrived on the system.

  User 1 Login: 2009:11:16 - 10:25:38

  User 2 Login: 2009:11:16 - 10:29:12

- The time line includes entries that are not within a time span that infers local creation.

A summary of the time line produced by Dataprocess.py is available in Figure 10.

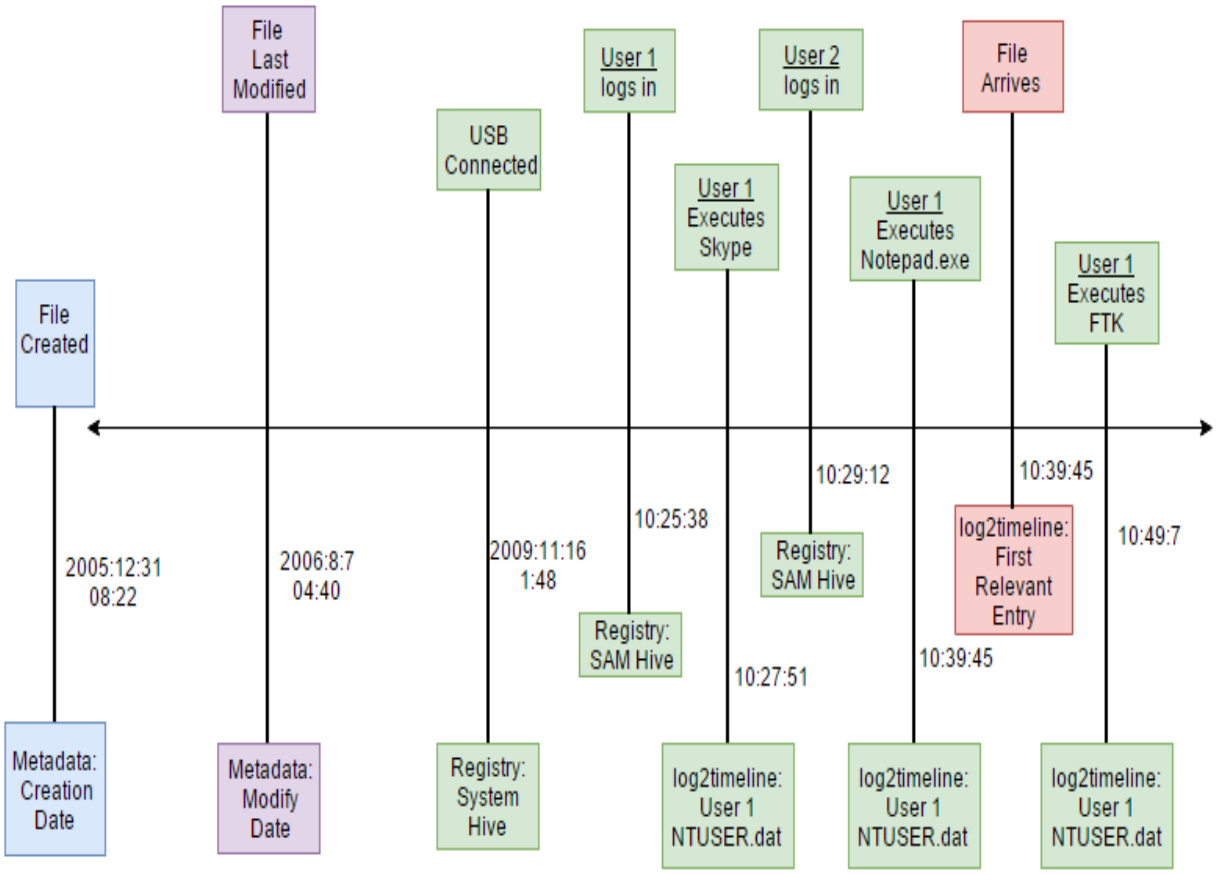


Figure 10: Time line Summary (Image 2: File 1)

The correlations presented show that the file could not have been created locally, as the file existed before the operating system was installed. The creation date is also much earlier than the file's first sighting on the system. The last editor was a user that does not exist on this system, meaning the file was not edited locally. Therefore, due to the editing flag and being active, we know that the file must have been copied and pasted to its current location. The user appears to have viewed the file using some office product based on the officeinteract flag. They most likely used excel based on the file type. The file appears to have arrived by

either USB or Skype based on the activated flags.

The time line further reinforces the notion that the file could not have been modified locally, as the file last modified date and time is long before the file arrival time. A USB device is connected on the same day as the file's arrival; however, this occurs 9 hours previous to the event of interest. Notepad is executed just before the file arrives, but it is unable to create an xls file. It was likely used to take notes based on the contents of the file of interest. This leaves Skype as the most likely source of the file. User 1 executes Skype within half an hour of the file's arrival. This is a reasonable time span for a conversation to take place, during which a file is transferred. The execution of FTK does not appear to be relevant to the file of interest. Therefore, User 1 most likely brought the file onto the system via Skype, moved it to its current location, viewed the contents using excel, and copied some of the information into a txt file.

Provenance Narrative

An external user edits the file after creation, but not locally. Someone creates the file on another system, and then User 1 transfers it to this system via Skype on 2009:11:16 at 10:39:45. Next, a user copies and pastes it to its current location, most likely User 1. This user then uses Excel to view the file.

Image 2: File 2

The following booleans are set to True for the second file of interest, a zip file, on image 2.

- **userinteract** - A user, user 1, interacted with the file of interest.
  Arrival/First Interaction: 2009:12:10 - 16:11:12

- **editing** - The file appears to have either been edited or copied within the system due its mtime being much different than its arrival time. The mtime is provided to the user.

– **samedaylogin** - User 3 has a last log in date that is the same day the file arrived on the system.

User 3 Login: 2009:11:16 - 15:06:02

– **ie5_content** - References to the file exist within Internet Explorer's Temporary Internet Files (TIF).

– The time line includes entries that are not within a time span that infers local creation.

A summary of the time line produced by Dataprocess.py is available in Figure 11.

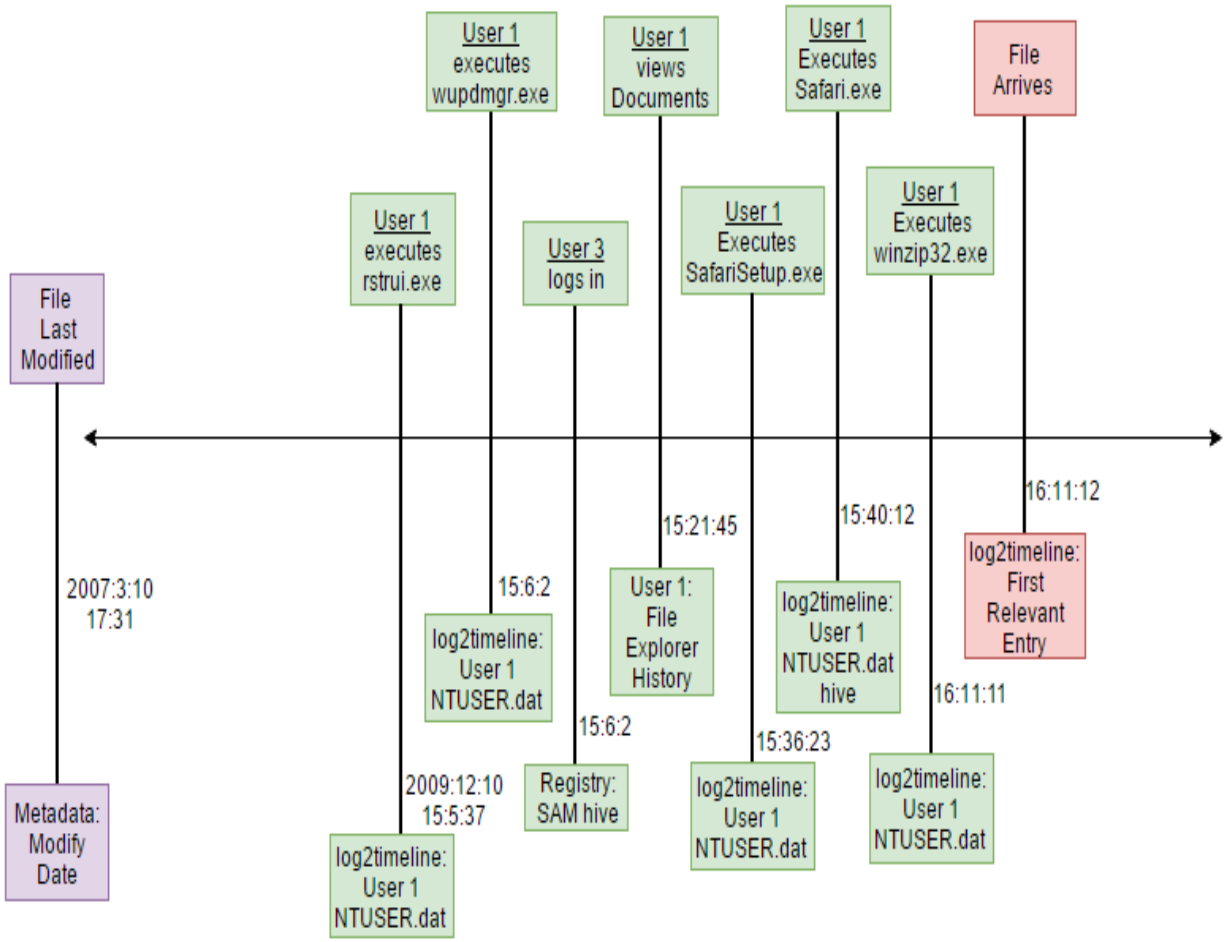


Figure 11: Time line Summary (Image 2: File 2)

A creation date is unavailable for this file, so the proper checks are unable to determine local creation. Looking at the time line; however, it is obvious that the file did not originate locally, as the file last modified date is long prior to that of the file's arrival. This date and time is pulled from the file contained within the compressed directory. From here, it is important to note that the ie5_content flag is activated. This points to a high probability of a web based file source. It is evident that user 1 most likely either copied or edited the file of interest. The editing is most likely the compressing of the file.

The time line reinforces and further refines the theory of a web source. The web browser, Safari, is installed and executed by user 1 20 minutes before the arrival of the file. Winzip is also executed by user 1 moments before the file is first seen on the system. This makes it highly likely that the file was downloaded by User 1 using Safari, then compressed and moved to its current location.

Provenance Narrative

Someone creates the file externally, and someone else modifies it on 2007:3:10 at 17:31. User 1 then downloads it via Safari on 2009:12:10 at 16:11:12. Next, User 1 compresses the file, and moves it to its current location.

Image 2: File 3

The following booleans are set to True for the third file of interest, an mp3 file, on image 2.

- **userinteract** - A user, User 1, interacted with the file of interest.
  Arrival/First Interaction: 2009:11:16 - 10:40:10

- **usbdatematch** - A removable disk was used on the same day that the file first arrived on system.
  USB Connected: 2009:11:16 - 1:48

– **skype30min** - Skype was used within 30 minutes of the file first being seen on system.

Skype Execution: 2009:11:16 - 10:39:45

– **samedaylogin** - User 1 and user 2 have a last log in date that is the same day the file arrived on the system.

User 1 Login: 2009:11:16 - 10:25:38

User 2 Login: 2009:11:16 - 10:29:12

– The time line includes entries that are not within a time span that infers local creation.

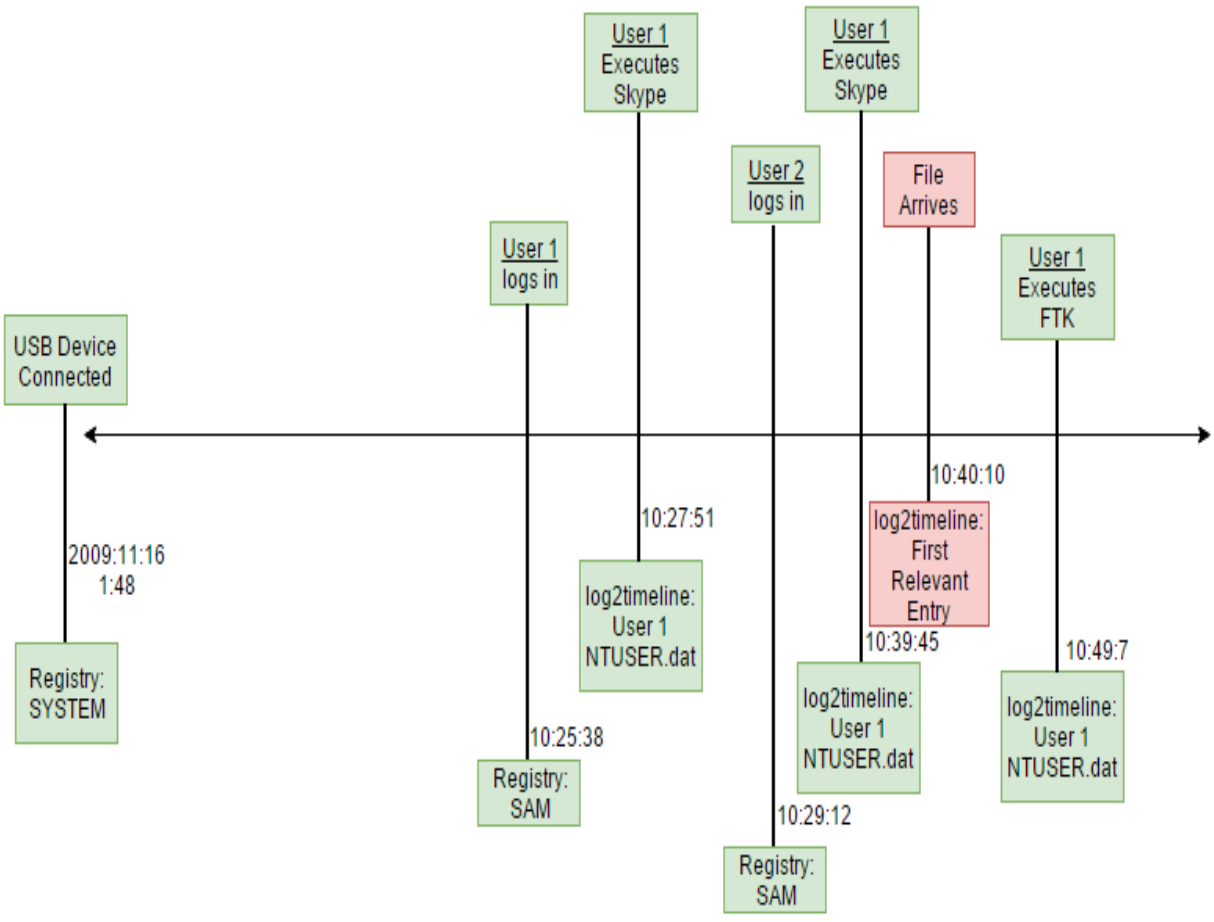


Figure 12: Time line Summary (Image 2: File 3)

Limited information is available about this file due to the decreased metadata associated with mp3 files compared to the other files analyzed. This results in only a few presented correlations, showing Skype usage within 30 minutes of the file's arrival, a USB connection on the day of its arrival, and two user log ins on the arrival day. Fortunately, the time line helps flesh out the remaining details necessary to determine how the file arrived on system. This time line looks very similar to the one shown for image 2: file 1. It appears that this file also arrived by Skype, most likely delivered along with File 1. It is also possible that this file, along with file 1, were transferred via USB. Due to the time difference between the USB connection and the file's arrival; however, Skype is a much more likely file source. The provenance available for this file is much more limited than what is achievable with Microsoft Office files.

Provenance Narrative

Someone creates the file externally, and User 1 acquires it via a Skype conversation on 2009:11:16 at 10:40:10.

Image 2: File 4

The following booleans are set to True for the fourth file of interest, a .doc file, on image 2.

- **userinteract** - A user, User 1, interacted with the file of interest.
  Arrival/First Interaction: 2009:11:16 - 7:36
- **usbdatematch** - A removable disk was used on the same day that the file first arrived on system.
  USB Connected: 2009:11:16 - 1:48
- **skypedatematch** - Skype was used on the same day that the file first arrived on the system.
  Skype Execution: 2009:11:16 - 10:39:45

– **samedaylogin** - User 1 and user 3 have a last log in date that is the same day the file arrived on the system.

User 1 Login: 2009:11:16 - 10:25:38

User 3 Login: 2009:11:16 - 15:06:02

– **officeinteract** - Office was used to interact with the file.

Arrival/First Interaction: 2009:11:16 - 7:36

– The time line includes entries that are not within a time span that infers local creation.

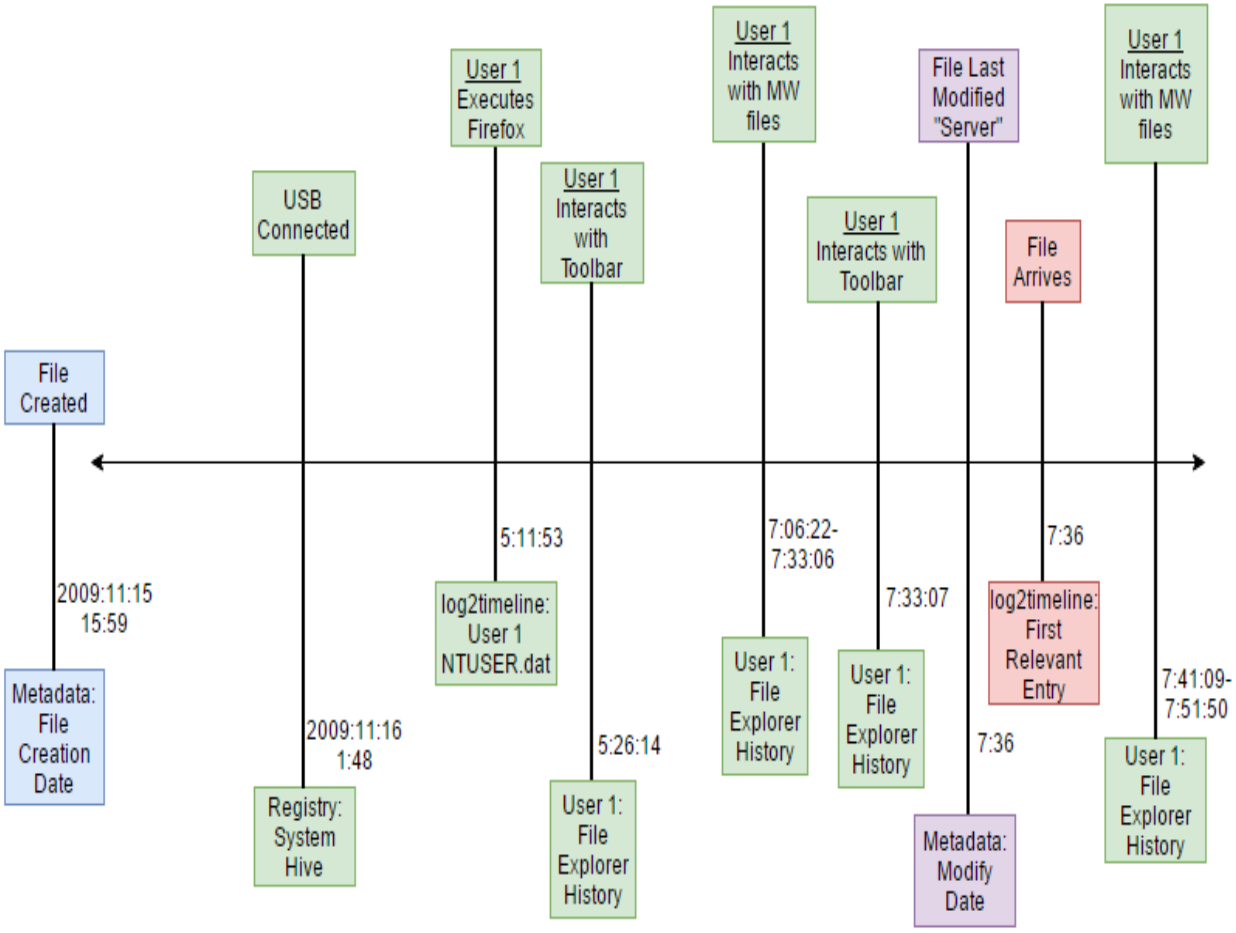


Figure 13: Time line Summary (Image 2: File 4)

The correlations show that user 1 interacted with the file of interest, and that they are the only user that interacted with the file. Therefore, user 1 allowed the file to arrive on the system. User 1 used an office tool, most likely Microsoft Word, to interact with the file after arrival, as the officeinteract flag shows. User 3 was logged on as well the day the file arrives, so they may have some knowledge of the file's arrival.

The time line entries do not infer local creation, as the file is created a full day before it arrives on the local system. so the file must have originated from an external source. A USB device was attached on the same day as the file's arrival, and Skype was used as well, marking them as potential sources for the file. However, upon looking at the time line, it immediately becomes apparent that Skype is not the source of the file, as it is executed after the file's arrival (Removed from time line due to being out of scope). The USB device is a possible source; however, Mozilla Firefox is the more probably file source based on it being executed in greater proximity to the file's arrival. Unfortunately, the system's Fire Fox history is erased, and therefore it is difficult to draw definitive conclusions about the origins of the file.

Some other important facts that help attribute the file to a browser source is that the name of the file's last modifier is simply "Server". This implies that the file was modified by some sort of web server before its arrival on the current system. This modification occurs moments before the file's arrival, and is therefore likely caused by an automated system. The user also interacts with many other .doc files during this time period, lending to the suspicion that they downloaded all of these file's within close proximity to one another using the Fire Fox web browser.

Provenance Narrative

Someone creates the file externally on 2009:11:15 at 15:59, and User 1 downloads it using Mozilla Firefox on 2009:9:8 at 4:50:50. User 1 then interacts with the file using Microsoft

Word some time near its arrival, although it is unclear exactly when.

Image 3: File 1

The following booleans are set to True for the first file of interest, a .jpg file, on image 3. Image 3 is a Windows XP machine with Service Pack 2 installed.

- **usbdatematch** - A removable disk was used on the same day that the file first arrived on system.
  USB Connected: 2009:9:8 - 15:27:45

- **ie5_content** - References to the file exist within Internet Explorer's Temporary Internet Files (TIF).
  Download Hosts Visited: 2009:9:8 - 2:50:27-3:12:43

- **userinteract** - A user, User 1, interacted with the file of interest.
  Arrival/First Interaction: 2009:9:8 - 4:50:50

When dealing with a .jpg file, there is limited metadata available. The creation date and time of the file is therefore unknown. Because of this, many common correlation checks are not possible. Fortunately, enough evidence is still present in this case to allow for an adequate rebuilding of the file's provenance. The first boolean, usbdatematch, shows that one or more USB devices were connected on the day of the file's arrival. Looking at the time line, it is apparent that these drives are not the source of the file, as they are both connected long after the file's arrival.

The next boolean, ie5_content, shows that the file is referenced within Internet Explorer's TIF. This occurrence has a strong correlation with the file arriving via a browser. The time line confirms this, showing that a user visited various download hosting websites about an hour prior to the file's arrival. The file has the [1] symbol appended to it, which means that it was downloaded twice. Therefore, a probably provenance of the file is that User 1

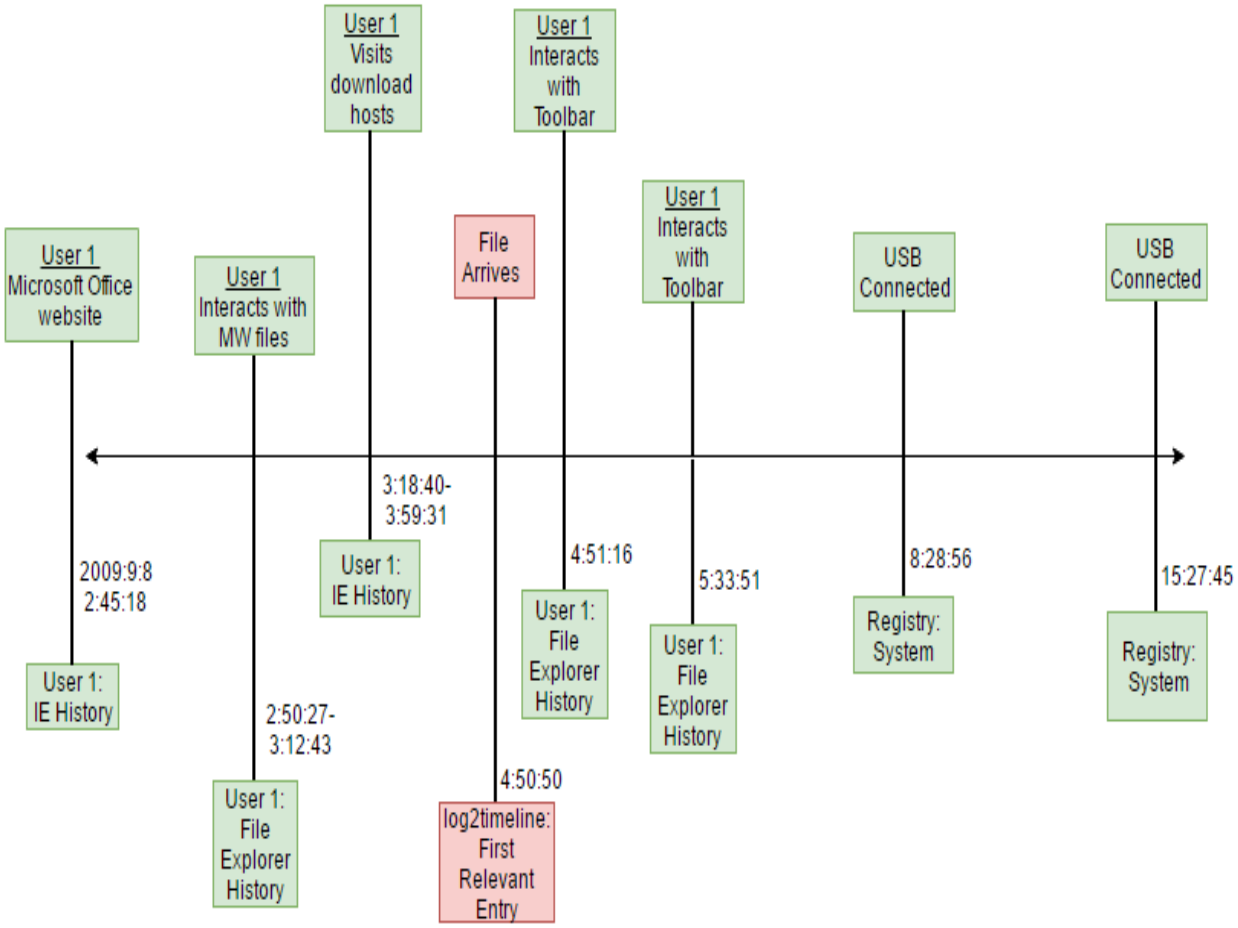


Figure 14: Time line Summary (Image 3: File 1)

downloaded the original file using the web browser at the last web download host noted in the time line. They then most likely walked away from the computer for an hour, forgot they downloaded the file, and downloaded it a second time, resulting in it arriving an hour after the user visited the source web link.

Provenance Narrative

An external user creates the file, and User 1 later downloads it via Internet Explorer on 2009:9:8 at 4:50:50.

Image 4

DataGather.py failed to gather the relevant information necessary to obtain the provenance on any of the file's on image 4. This is due to log2timeline being unable to scan the source due to one of its library's, libsigscan, inability to read a necessary buffer. Joachim Metz, the creator of log2timeline, responds to this error [22] but no fix is applied at this time.

Image 5: File 1

The following booleans are set to True for the first file of interest, a .jpg file, on image 5. Image 5 is a Windows XP machine with Service Pack 2 installed.

- **userinteract** - Two users, Users 1 and 2, interacted with the file of interest.
- **ie5_content** - The file is referenced within Internet Explorer's Temporary Internet Files (TIF).

Unfortunately, not enough data is available on this file in order to reliably build a time line. There is also insufficient metadata available to determine when the file was created, or if it was modified at any point post-creation. References to the file exist within Internet Explorer's TIF; however, which means that it is almost certain the final arrived via a browser. It is also apparent that two user's on the system, user's 1 and 2 interacted with the file post-arrival.

Image 5: File 2

The following booleans are set to True for the second file of interest, a .doc file, on image 5.

- **userinteract** - A user, Users 2, interacted with the file of interest.
  Arrival/First Interaction: 2009:12:18 - 16:15:00
- **Creator not Modifier** - Looking at the file's metadata, the software sees that the file's creator is not the same user as the file's modifier. This is not a boolean flag, but the software still outputs this information. The modifier's name is also provided.

– **ie5_content** - References to the file exist within Internet Explorer's Temporary Internet Files (TIF).

Sports Site Visit: 2009:12:18 - 16:01:27

– **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.

Creation: 2009:12:18 - 16:15:00

Arrival: 2009:12:18 - 16:15:00

– **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

Creation: 2009:12:18 - 16:15:00

Arrival: 2009:12:18 - 16:15:00

– **officeinteract** - Office was used to interact with the file.

Arrival/First Interaction: 2009:12:18 - 16:15:00

The results of running the software on this image/file show that the file was most likely downloaded by user 2. A reference to the file is found within the TIF folder of Internet Explorer, which correlates to the file arriving from a web source. It is also apparent, based on the time line, that the user visited a variety of websites within close proximity to the file's arrival that may have been the source of the file. The most likely of these websites is the sports website.

Based on the date check and time check flags, it is apparent that the file was created at the same time that it arrived on the system. This could be due to the user using some web based program to generate the file, and then downloading it upon generation. The file is modified a year later based on its metadata. Unfortunately, it is not at all clear how this occurred. The only program executions that occur within close proximity to this

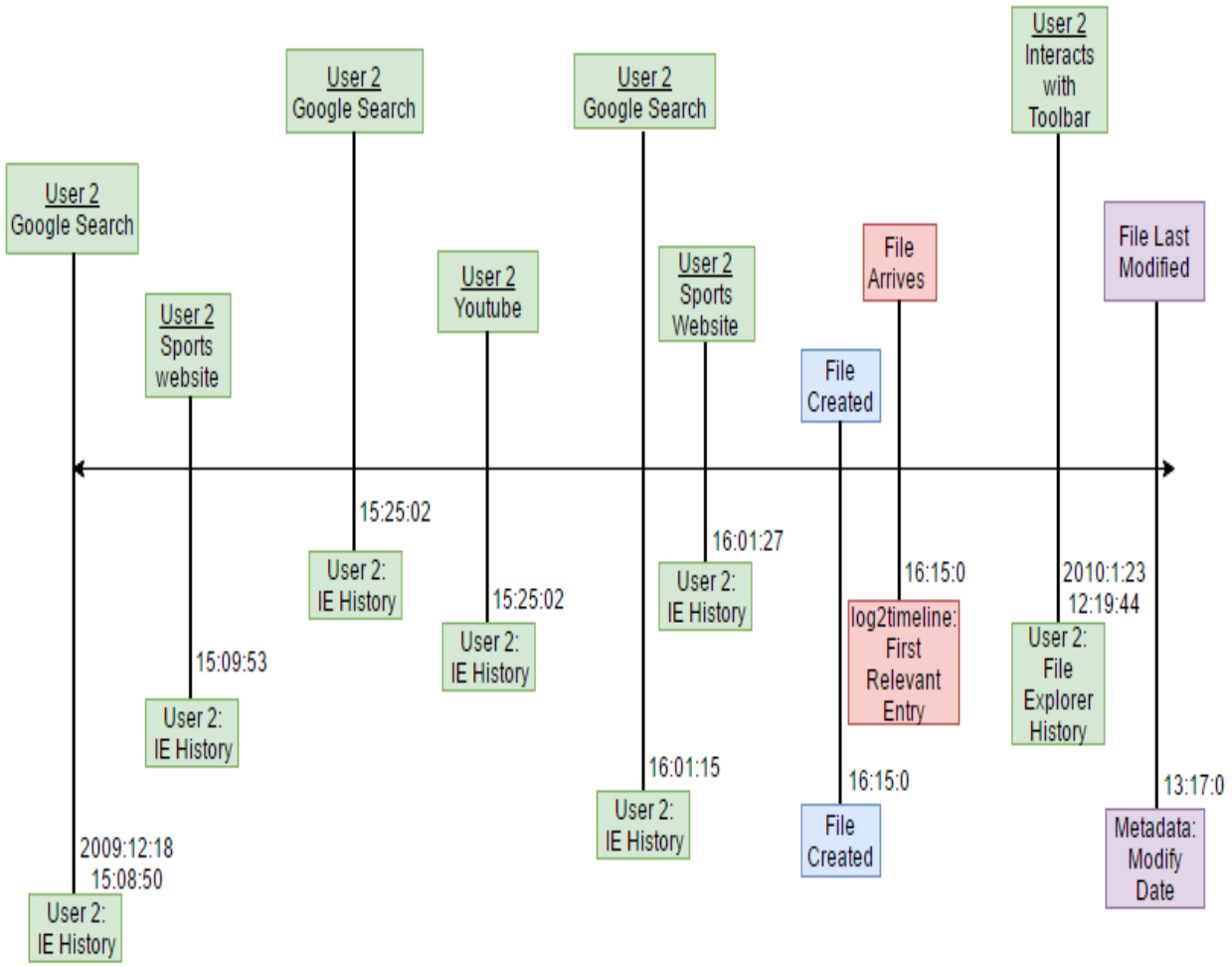


Figure 15: Time line Summary (Image 5: File 2)

modification is user 2 opening the Windows toolbar. The user name that last modified the file also does not exist on the system, creating further confusion.

Provenance Narrative

A sports website creates the file on 2009:12:18 at 16:15:00, and User 2 downloads it immediately afterward using Internet Explorer. An external user somehow modifies the file on 2010:1:23 at 13:17:0.

Image 6: File 1

The following booleans are set to True for the first file of interest, a .docx file, on image

6. Image 6 is a Windows XP machine with Service Pack 3 installed.

   - **userinteract** - A user, Users 1, interacted with the file of interest.
   - **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.
   - **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.
   - **officeinteract** - Office was used to interact with the file.

Unfortunately, there wasn't enough information available on this file to properly generate a time line. The only things that are valuable within the time line, are the creation, modification, and arrival times. This information still provides valuable input, as all of these dates/times are the same. This greatly increases the probability that the file was locally created. The activated boolean variables help confirm these suspicions. It is evident that a user interacted with the file of interest. The investigator is also able to see the file arrived within 30 minutes of its creation, and that a user, obviously the user who interacted with the file, used office to do something with the file. Based on all of these factors, user 1 used Microsoft Word to create the file of interest.

Image 6: File 2

The following booleans are set to True for the first file of interest, a .docx file, on image 6. Image 6 is a Windows XP machine with Service Pack 3 installed.

   - **userinteract** - A user, Users 1, interacted with the file of interest.
   - **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.

– **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

– **officeinteract** - Office was used to interact with the file.

This file presents a similar situation to the one previous to it. Considering it was found in the same directory, and that the same boolean variables are active, the file most likely originated from the same source. User 1 most likely created this file, just as they created file 1.

Image 7: File 1

The following booleans are set to True for the first file of interest, a .doc file, on image 7. Image 7 is a Windows XP machine with Service Pack 2 installed.

– **userinteract** - A user, User 1, interacted with the file of interest.

– **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.

– **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

– **officeinteract** - Office was used to interact with the file.

– **time linerelevant_removable_disk_usage** - A time line entry references the file of interest, as a well as a USB drive.

– **systemuser** - The file's creator has the same user name as a user who exists on this system.

– **systemmod** - The file's last modifier has the same user name as a user who exists on this system.

There was not enough information to produce a time line, but enough conclusions are drawn from the activated boolean variables to determine the source of the file. It is immediately apparent that a user on the system interacted with the file of interest. Looking at the rest of the output, it is visible that this user is the same one who created and last modified the file. This is apparent thanks to the systemuser and systemmod flag's activation. It is also evident that an office tool was used to interact with the file of interest.

A USB device is in the creating user's recently used documents along with this file, which often is a symptom of a USB source for the file. In this case, the overwhelming evidence for local creation in this case overshadows this one USB factor. Based on the factors presented, it is apparent that user 1 created the file on the system, and then modified it half an hour later. The later modification time is visible based on the few obtained time line entries.

Image 7: File 2

The following booleans are set to True for the first file of interest, a .doc file, on image 7.

- **userinteract** - A user, User 1, interacted with the file of interest.
- **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.
- **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.
- **officeinteract** - Office was used to interact with the file.
- **time linerelevant_removable_disk_usage** - A time line entry references the file of interest, as a well as a USB drive.
- **systemuser** - The file's creator has the same user name as a user who exists on this system.

– **systemmod** - The file's last modifier has the same user name as a user who exists on this system.

The booleans activated for this file are the same as the one previous, with the same user as the source. Therefore, it is reasonable to draw the conclusion that user 1 locally created this file as well.

Image 7: File 3

The following booleans are set to True for the first file of interest, a .doc file, on image 7.

– **userinteract** - A user, User 1, interacted with the file of interest.

– **word_modify_day** - Microsoft word was used on the same day as this file's last modification.

– **date_check** - Relevant time line entries exist that refer to the file of interest on the same day that it was created.

– **time_check** - Relevant time line entries exist that refer to the file of interest within thirty minutes of its creation.

– **officeinteract** - Office was used to interact with the file.

– **time linerelevant_removable_disk_usage** - A time line entry references the file of interest, as a well as a USB drive.

– **systemuser** - The file's creator has the same user name as a user who exists on this system.

– **systemmod** - The file's last modifier has the same user name as a user who exists on this system.

While this file's provenance is almost the same as the one's previous to it, and interested differentiation occurs. The word_modify_day flag activates, and the program includes the date and time of a Microsoft word execution with the time line. This shows that MW was executed locally within 5 minutes of the file's last modification date, by user 1. All other factors are the same, which means that this simply helps reinforce the notion of local creation and modification, and that the user, user 1, was well aware of the presence of this file.

4.3 Summary

In the majority of the test cases presented, the AutoProv successfully rebuilt the file's provenance. The controlled testing shows that all of the implemented boolean variables behave as expected. Within the context of the RDC, they point out valuable correlations that show the examiner likely sources of the file. When combined with the time line, if available, the source of the file is almost always evident, as well as much of the activity surrounding the arrival of the file.

Unfortunately, the data within images 6 and 7 did not allow for a thorough and complete time line. This was due to a lack of file explorer history, and no obvious program executions within the NTUSER.dat hive surrounding the arrival of the file. It is possible that this is due to purposeful obfuscation, which this program is not designed to detect or circumnavigate.

5. Conclusions

It is possible to rebuild most of a file's provenance on a Windows forensic image. This task is far more difficult than it is on a live machine, as many of the resources available when dealing with a live platform are not present on a forensic image, including the system's memory and the live tracking of user activities. Despite these limitations, there is more often than not enough information available in the remaining provenance sources to recreate the various activities related to a file of interest.

The metadata of select files provides essential data for provenance recreation. Microsoft Office files, such as .xls and .docx files, often have highly useful metadata. In contrast, the availability of this information is much more limited in other files, such as .mp3. This data includes who created the document, when it was created, who last modified the document, when that modification occurred, and when the document was last accessed. By combining this information with the rest of the data gathered from within the system, provenance accuracy is greatly increased. Therefore, provenance recreation is far more feasible when dealing with a file that has detailed metadata available.

The registry hive contains many lucrative sources of information that aid in provenance recreation. Primarily, the NTUSER hives for the various non-default users present on a system contains a plethora of valuable data. Various program executions and other activities that lead to the creation and/or arrival of a file on the system are found within this registry hive. Compiling entries that occur within a relevant time span, and parsing them for possible correlations yields many positive results that often enables the user to determine the file's source, and other interactions with the file that are relevant to its provenance.

One of the most unexpectedly valuable sources of file provenance was the Internet Explorer history directories. These directories and files were originally parsed to gather information on the users browsing activity. Upon closer inspection, it became apparent that they

also contain information on all of the user's file explorer browsing activities as well. This information helped draw more conclusive provenance in many cases where a file's metadata was not quite as thorough.

In tandem with this, the various other browser histories often helped make it abundantly apparent when a file was downloaded from a web source. For example, there was several instances where the web address of a mail server was referenced just prior to the arrival of a file. This makes it apparent that the user most likely acquired the file from the web server, downloading it onto the system of interest.

Various other regions of the registry were explored as well, and found useful for determining the provenance of a file. For example, the system hive contains information on the system's installation date, as well as various USB interactions. This helped to determine when a file was created long before the installation of a system, lending to the conclusion that the file could not have been created locally. The USB interactions, when in close proximity to a file's arrival on system, helped to determine whether a file could have possibly been transferred from a referenced USB device.

5.1 Future Work

The metadata of many of these files was used to help determine their provenance. Unfortunately, it is possible for a savvy user to easily modify these values, obfuscating the true provenance. In the worst case, these alterations could lead to the incrimination of a unrelated party. Methods to detect these alterations, and notify the user of their occurrence, would be useful in ensuring the accuracy of the provenance presented.

There are additional methods that lead to the obfuscation of a file's provenance. A powerful tool that is briefly mentioned earlier in this thesis is Timestomp. This tool alters many of the values that this thesis uses to determine the provenance of a file. Any methods of detection that aid a user's awareness of the use of this software are valuable. Detection of

the use of Timestomp is a highly difficult proposition, and therefore it was outside the scope of this thesis.

There are many ways a file arrives on a system, and even more that it is interacted with. This thesis barely touches on the surface of these possibilities, seeking only to provide a proof of concept that it is possible to recreate a file's provenance if enough possible interaction and arrival sources are accounted for. The automated browser history investigation that this software accomplishes is useful for determining when web email services are used. Unfortunately, it does not account for the use of software such as Outlook to acquire emails and their respective attachments. An excellent addition to this project would therefore be the automated detection of various Outlook interactions, and any history associated with outlook itself.

Another possible avenue of arrival is FTP. There is a variety of FTP software that are capable of transferring a file from one machine to another. Exhaustive accounting for each of these pieces of software would be arduous, therefore finding some sort of common denominator between files that arrive by this methodology is essential. Finding this common sign within the registry or some similar location may allow the software to automatically determine when FTP accommodates a file's arrival.

A similar problem this thesis encountered was detecting when torrenting software is used to acquire a file. The method presented in the methodology shows, using FrostWire as an example, that it is possible to exhaustively account for the software used to obtain a file. Modifying this method into something that is more general would be highly valuable for ensuring similar software works in many cases, and that it remains viable as future torrenting software is introduced.

This software uses the MAC times in order to determine when activities such as file movement occur. These values are modifiable, and therefore a more resilient method of determining when movements, cut-paste, and accesses occur would be beneficial to the func-

tionality of AutoProv. For example, finding another sign of these activities present in the registry or some other relevant location, and comparing this with the MAC values present on the file of interest, could lead to interesting insights into whether value tampering occurred.

Appendix A: Abbreviations

DFRWS	Digital Forensic Research Workshop
DOJ	Department of Justice
EXIF	Exchangeable Image File Format
FPMS	File Provenance Maintenance System
FTK	Forensic Tool Kit
FTP	File Transfer Protocol
MAC	Modification, Access, Change
MW	Microsoft Word
PVM	Provenance Versioning Model
RAM	Random Access Memory
RDC	Real Data Corpus
SAM	Security Account Manager
TSK	The Sleuth Kit
USB	Universal Serial Bus
WAP	Wireless Access Point

References

[1] N. Balakrishnan, T. Bytheway, L. Carata, O. Chick, J. Snee, S. Akoush, R. Sohan, M. Seltzer, A. Hopper, "Recent advances in computer architecture: the opportunities and challenges for provenance" *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*, 2015.

[2] N. Beebe, G. Dietrich, "A new process model for text string searching." *IFIP International Conference on Digital Forensics* Springer, NY, 2007.

[3] F. Buchholz, C. Falk, Design and implementation of Zeitline: a forensic time line. *Proceedings of the 2005 Digital Forensic Research Workshop* New Orleans, LA, 2005.

[4] B. Carrier, E. Spafford, "Automated digital evidence target definition using outlier analysis and existing evidence. In Proceedings of the" 2005 Digital Forensics Research Workshop

[5] O. Carroll, S. Brannon, T. Song, "Computer Forensics: Digital Forensic Analysis Methodology" *Computer Forensics Volume 56 Number 1*, 2008

[6] H. Carvey, Windows Registry Forensics: *Advanced Digital Forensic Analysis of the Windows Registry* Syngress, Massachusetts, 2016.

[7] A. Case, A. Cristina, L. Marziale, G. Richard, V. Roussev, FACE: automated digital evidence discovery and correlation. *Digital Investigation Volume 5*, pp. 65–75, 2008

[8] E. Casey, *Digital evidence and computer crime: Forensic science, computers, and the internet* pp. 3-34, 2011.

[9] D. Edwards, *Computer forensic time line analysis with tapestry* 2011

[10] Unix Time Stamp (http://www.unixtimestamp.com/)

[11] J. Farrell, *A Framework for Automated Digital Forensic Reporting*, Naval Postgraduate School, 2009

[12] K. Gudjonsson, "Mastering the Super time line With log2timeline" *SANS Institute Information Security Reading Room*, 2010

[13] P. Harvey, Exiftool (http://owl.phy.queensu.ca/phil/exiftool/), 2012

[14] R. Hasan, R. Sion, and M. Winslett, "Preventing history forgery with secure provenance". *ACM Transactions on Storage (TOS) Volume 5 Issue 4*, 12:1–12:43, 2009.

[15] C. Jensen, H. Lonsdale, E. Wynn, J. Cao, M. Slater, T. Dietterich, "The life and times of files and information: a study of desktop provenance" *SIGCHI Conference on Human Factors in Computing Systems Proceedings*, pp. 767-776, 2010

[16] C. Lee, J. Trost, N. Gibbs, R. Beyah, J. Copeland "Visual Firewall: Real-time Network Security Monitor" *IEEE Visualization for Computer Security*, 2005

[17] "Linux Howtos and FAQs." LINUX-FAQS.INFO. Web.

[18] D. Manson, A. Carlin, S. Ramos, A. Gyger, M. Kaufman, and J. Treichelt, "Is the open way a better way? Digital forensics using open source tools," *System Sciences HICSS 40th Annual Hawaii International Conference on. IEEE*, pp. 266b, 2007

[19] D. Margo, M. Seltzer, "The Case for Browser Provenance." *Workshop on the Theory and Practice of Provenance*, 2009.

[20] D. Margo, R. Smogor Using Provenance to Extract Semantic File Attributes *Proceedings of the 2nd conference on Theory and practice of provenance*, pp. 7–7, 2010

[21] S. Garfinkel, P. Farrell, V. Roussev, G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora" *Proceedings of the 9th Annual Digital Forensic Research Workshop (DFRWS)*, 2009

[22] log2timeline unable to read buffer (https://github.com/log2timeline/plaso/issues/869) 2016

[23] Merriam-Webster Dictionary (https://www.merriam-webster.com/dictionary/provenance)

[24] W. Minnaard, *MSc final research project report* Timestomping NTFS University of Amsterdam, 2014

[25] K. Muniswamy-Reddy, D. Holland, U. Braun, M. Seltzer, "Provenance-Aware Storage Systems" *USENIX Annual Technical Conference Refereed Paper*, 2006

[26] Nirsoft ChromeHistoryView Tool http://www.nirsoft.net/utils/chrome_history_view.html

[27] NirSoft MZHistoryView Tool (http://www.nirsoft.net/utils/mozilla_history_view.html), 2007

[28] Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Andy Hopper, "OPUS: A Lightweight System for Observational Provenance in User Space", *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance, Berkeley, CA, USA, series TaPP '13* pp. 8:1–8:4, 2013.

[29] C. Rusen, "Simple questions: What is a file's metadata and how to edit it in Windows?" *Digital Citizen*, (http://www.digitalcitizen.life/what-file-s-metadata-and-how-edit-it), 2016

[30] S. Sultana, E. Bertino, A File Provenance System *CODASPY*, 2013

[31] SANS Forensic Artifact 6: UserAssist (http://sploited.blogspot.com/2012/12/sans-forensic-artifact-6-userassist.html), 2012

[32] M. Willard, "Getting the Most out of your Firewall Logs" *SANS Institute InfoSec Reading Room*, 2002

[33] E. Zadok, I. Badulescu, A Stackable File System Interface For Linux *LinuxExpo Conference Proceedings*, 141-151, 1999

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
23-03-2017	Master's Thesis	Sept 2015 - March 2017

4. TITLE AND SUBTITLE

AutoProv: An Automated File Provenance Collection Tool

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Good, Ryan A, 2nd Lt USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DoD Cyber Crime Center (phone: 410-694-4310)
Eoghan Casey (email: Eoghan.casey@dc3.mil)
1190 Winterson Rd
Linthicum, MD 21090

10. SPONSOR/MONITOR'S ACRONYM(S)

DC3

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

A file's provenance is a detailing of its origins and activities. There are tools available that are useful in maintaining the provenance of a file. Unfortunately for digital forensics, these tools require prior installation on the computer of interest while provenance generating events happen. The presented tool addresses this by reconstructing a file's provenance from several temporal artifacts. It identifies relevant temporal and user correlations between these artifacts, and presents them to the user. A variety of predefined use cases and real world data are tested against to demonstrate that this software allows examiners to draw useful conclusions about the provenance of a file.

15. SUBJECT TERMS

Windows Forensics, forensic timelines, file provenance

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert L. Peterson AFIT/ENG
U	U	U	UU	98	19b. TELEPHONE NUMBER *(include area code)* 937-255-3636 x 4281 Gilbert.peterson@afit.edu